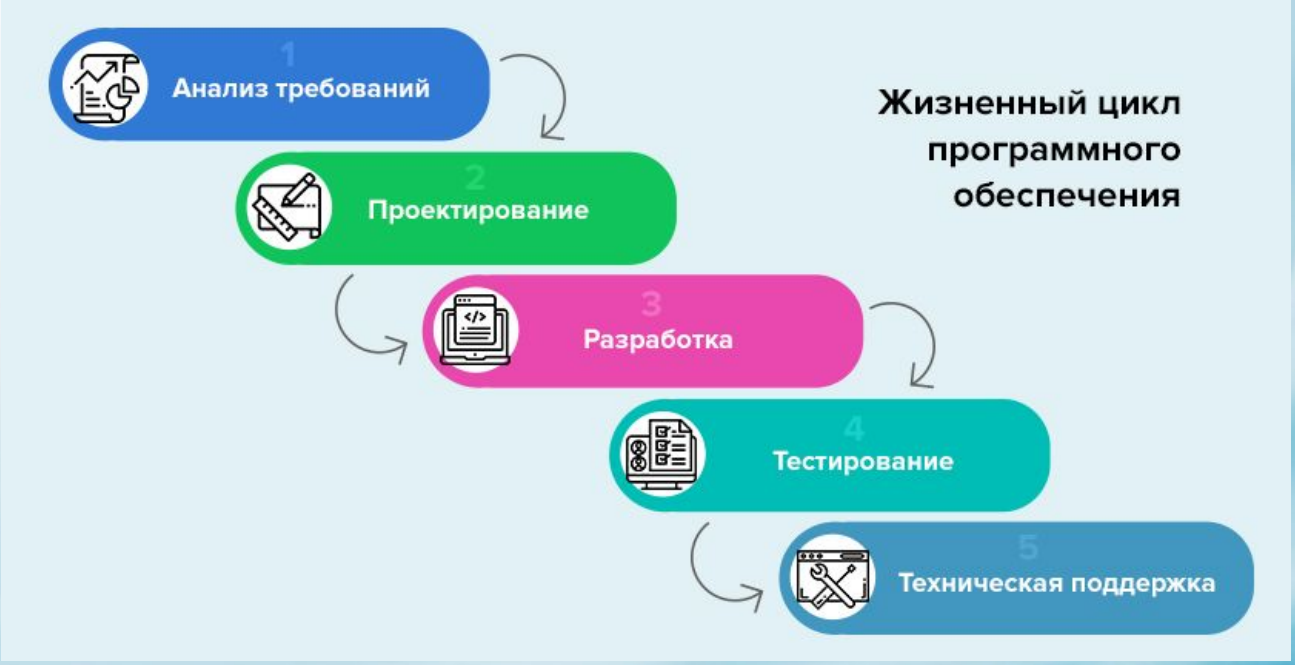
УВО «Университет управления «ТИСБИ»

ПРОГРАММИРОВАНИЕ

Ст. преподаватель: Якунина Е.А.

Казань 2021

Тема 1. **Этапы создания программного обеспечения**



При разработке сложного ПО используют одну из двух технологий:

1. Структурное программирование
2. Объектно-ориентированное программирование

При структурном подходе процесс разработки программ можно разделить на следующие этапы:

1. *Постановка задач*
2. *Анализ*
3. *Проектирование*
4. *Реализация*
5. *Модификация*

Постановка задачи

Постановка задачи включает в себя:

1. *Описание целей*
2. *Описание исходных данных*
3. *Описание особых ситуаций*
4. *Описание требований к используемому ПО и техническому обеспечению*

Анализ и формальная постановка

На данном этапе по результатам анализа задачи строят математическую модель ее решения и определяют метод требования исходных данных в результат

Пример: найти площадь прямоугольника по заданным длинам сторон.

a – число

b – число

S=a*b

Проектирование

Различают 2 вида проектирования:

- логическое;
- физическое.

Логическое проектирование предполагает детальную переработку последовательности действий будущей программы

При физическом проектировании осуществляют привязку разработанного алгоритма к имеющему набору технических и программных средств.

Реализация

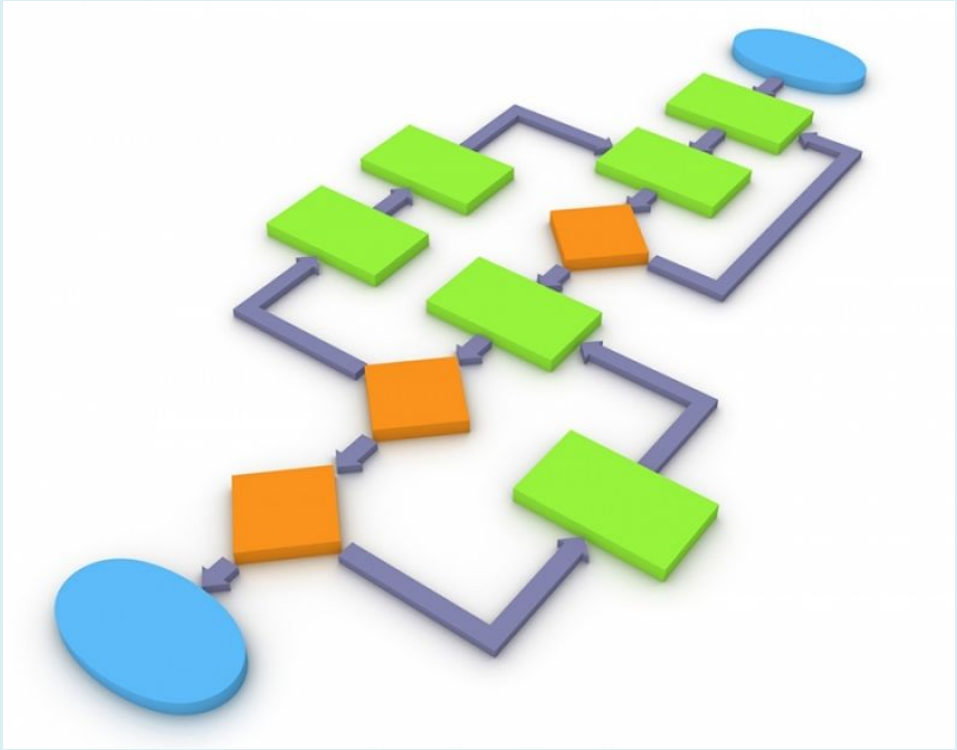
1. Составление текста программы с использованием конкретного языка программирования в соответствии с разработанным алгоритмом
2. Ввод текста программы в компьютер
3. Перевод текста программы в последовательность машинных команд (кодов). Этот этап называется **компиляцией (или трансляцией)** программ. Выполняется при помощи специальных программ **трансляторов и компиляторов**.
4. В процессе компиляции могут возникнуть ошибки, которые называются *синтаксическими*. В этом случае компилятор аварийно завершает выполнение программы, выдав сообщение об ошибке.
5. На данном этапе происходит объединение отдельных фрагментов программы в единую программу, этот процесс называется **компоновкой**. Программа – *компоновщик.*
6. В процессе выполнения программы запрашивается ввод исходных данных, осуществляется их обработка и выводится результат. Пример, деление на ноль, обращение к несуществующим данным.
7. Такие ошибки называются ошибками выполнения, чтобы их устранить, используется процесс отладки программы.

Модификация

Причинами новых версий может являться:

- Необходимость исправления ошибок, выявленных в процессе длительной эксплуатации
- Изменение среды
- Процесс совершенствования (улучшение интерфейса и увеличение набора выполняемых функций)

Тема 2. **Понятие об алгоритмах**



Алгоритм – четкая определенная последовательность действий (шагов), которые приводят к выполнению задачи за конечное число шагов

Алгоритм должен отвечать определенным требованиям:

1. *Наличие исходных данных*
2. *Наличие вывода результата*
3. *Однозначность*
4. *Общность*
5. *Корректность*
6. *Конечность*
7. *Эффективность*

Любой алгоритм может быть составлен с использованием следующих последовательных действий:

- *Линейная последовательность*
- *Разветвленная последовательность*
- *Циклическая последовательность*

Линейная структура предполагает, что каждое действие для достижения результата выполняется последовательно друг за другом.

Разветвленная структура предполагает, что выполнение действий зависит от значений одного или нескольких параметров (или от выполнения условий)

Циклическая структура предполагает, что для получения результата, некоторое действие необходимо выполнить несколько раз.

Любой алгоритм можно описать с использованием одну из следующих форм:

- *Словесное описание*
- *Графическое описание (блок-схемы)*
- *С использованием псевдокодов*

Словесное описание

$a*x^2 + b*x + c = 0$

1. **Исходные данные** – коэффициенты *a, b, c*
2. **Промежуточные данные** $D = b^2 - 4*a*c$

   $D < 0$ – корней нет

   $D = 0$ – 1 корень $x = -\frac{b}{2*a}$

   $D > 0$ – 2 корня $x = -\frac{b \pm \sqrt{D}}{2*a}$

1 Шаг: ввести исходные значения *a, b, c*

2 Шаг: вычислить значение дискриминанта по формуле *a, b, c*

3 Шаг: проверить , если $D < 0$, то вывести сообщение корней нет $x = -\frac{b}{2*a}$

4 Шаг: если $D = 0$, то найти единственный корень по формуле

5 Шаг: если $D > 0$, то найти оба корня по соответствующим формулам $x = -\frac{b \pm \sqrt{D}}{2*a}$

6 Шаг: вывести результат

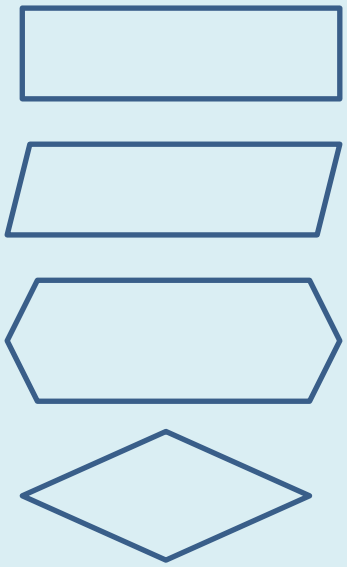
В данном алгоритме не учтены особые ситуации, связанные с вводом исходных значений-коэффициентов и которые могут привести к возникновению ошибок:

Алгоритм:

1. Вводим *a, b, c*
2. Если *a=0, b=0, c =0*, то вывести соответствующее сообщение
3. Если *a=0, b≠0 и c≠0,* то найти $x = -\frac{c}{b}$
4. Если *a≠0*, то вычисляем дискриминант *D*
5. Далее по алгоритму

Графическое описание

При описании алгоритма с помощью блок-схем, каждое действие обозначают блоком особой формы.



- действие

- ввод, вывод

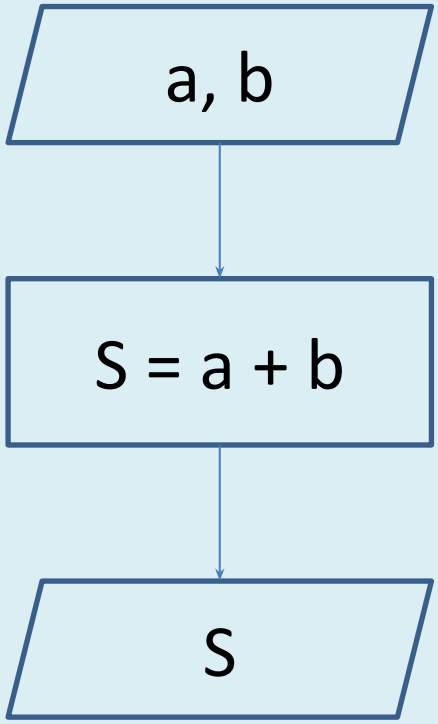
- цикл

- условие

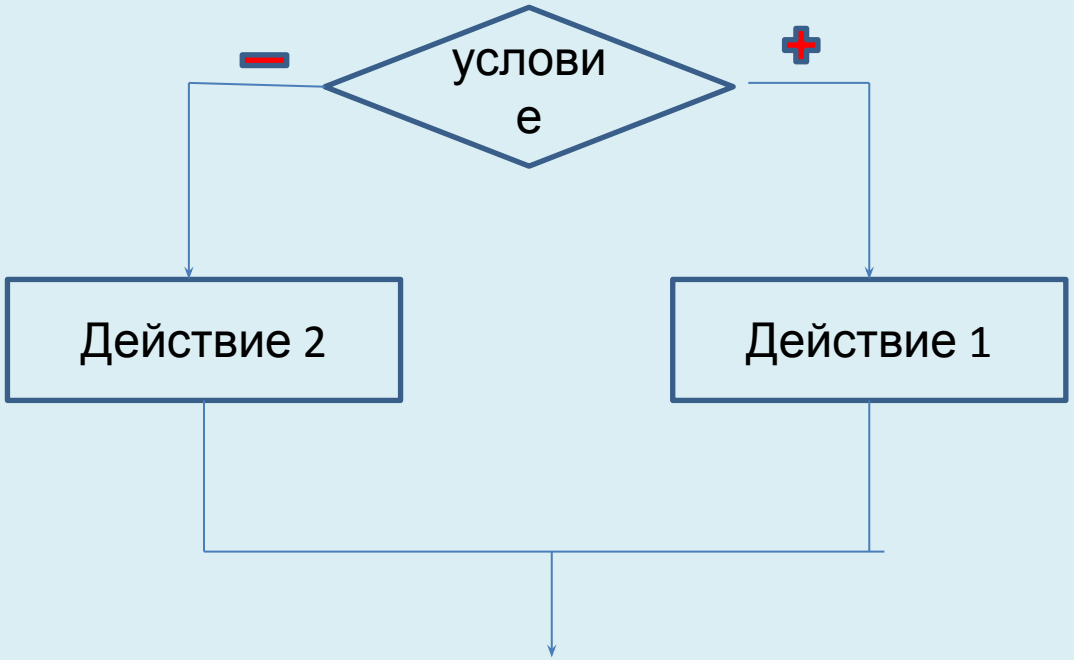
Любой сложный алгоритм описывается с помощью 3-х основных направляющих структур:

1. *Следование (линейное)*
2. *Проверка условий (разветвленное)*
3. *Циклическое*

Следование

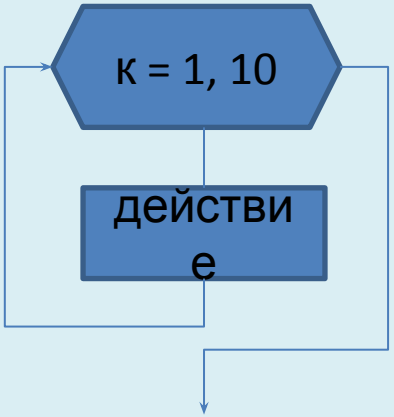


Проверка условий



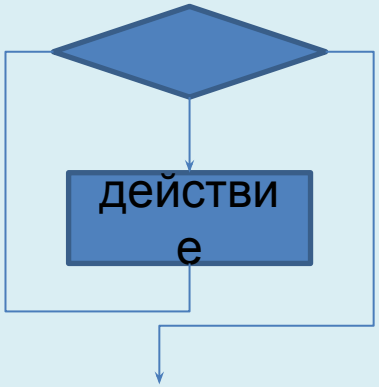
Циклическая структура

А) Цикл с известным числом повторений

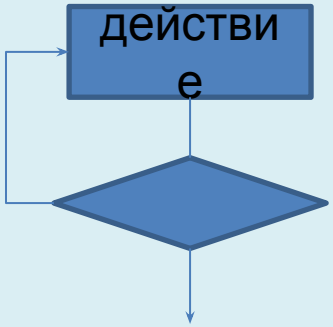


Б) Цикл с неизвестным числом повторений. В данном случае количество повторов зависит от выполнения или невыполнения заданного условия

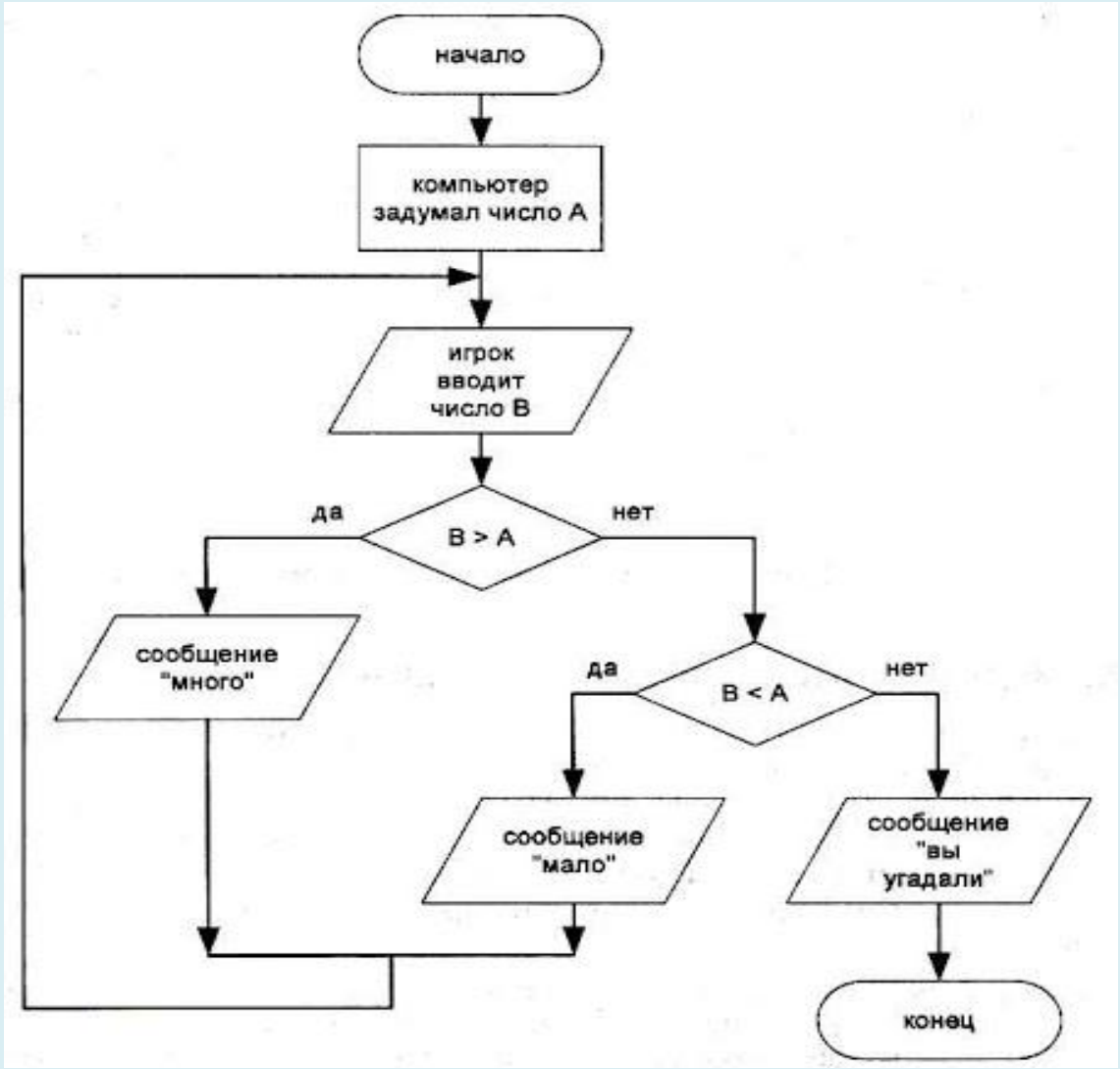
- Цикл с предусловием



- Цикл с постусловием

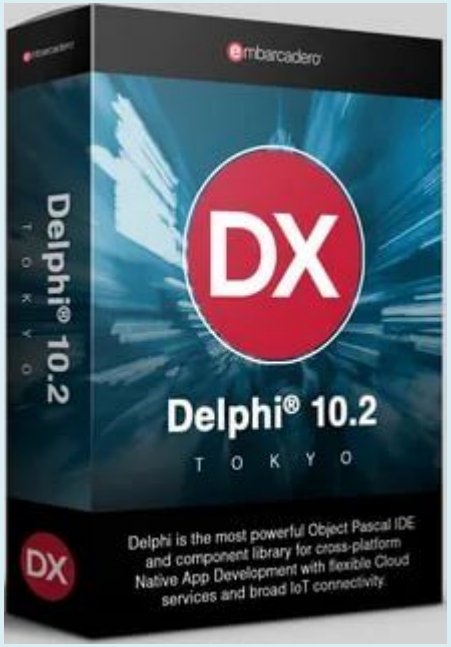


Псевдокод – описание алгоритма, который базируется на тех же управляющих структурах, но для каждых структур используют свою форму описания.



Тема 3. Общая структура программы на языке Delphi (Паскаль)

https://lazarus-rus.ru/download/



Синтаксис – это совокупность правил, определяющих допустимые конструкции языка, т.е. форму.

Семантика – это совокупность правил, определяющих смысл синтаксически корректных записанных конструкций языка, т.е. его содержания.

Семантику языка программирования закладывают в его компилятор, т.о. синтаксически грамотно записанные программы после преобразовываются последовательно в машинные коды, обеспечивает выполнение компьютером необходимых действий.

Описание синтаксиса включают определение алфавита и описание различных допустимых конструкций языка из символов алфавита и более простых конструкций.

Язык программирования Delphi

1) Латинские буквы (*заглавные и прописные*)
2) Наборы цифр от *0 до 9*
3) Специальные знаки: =, >, <, >=, <=, <>, *, /, ;, :, ‘ ‘, [], (), ...
4) Специальные служебные слова, написание которых должно полностью соответствовать требованиям языка: *begin, and, for, while, array* ...

При написание программы применяются:

- **Константы**
- **Переменные** - это данные, значение которых могут изменяться при выполнении программы
- **Выражения –** это константы или переменные, объединены каким-либо знаком операций
- **Операторы** – это специальные символы и слова, выполняющие действия
- **Функции, процедуры и модули** – это отдельные программные модули, имеющие свои собственные имена и подключающие к основной программе

Для использования в программе входных и выходных данных, промежуточных данных вводятся текстовые обозначения, соответствующих переменных. Эти обозначения называются **именами** или **идентификаторами**. Все действия в программе осуществляются с помощью соответствующих переменных. Переменные заменяют область памяти компьютера, где при выполнении программы находятся соответствующие значения.

Любая программа на языке Delphi состоит из 3-х частей:

1 часть – *заголовок программы*

2 часть – *раздел описания*

3 часть – *раздел выполнения действий*

I. Заголовок программы

Program *имя_программы;*

Program Primer1;

II. Раздел описания

Описание константы

Const – *константа*

```
Const pi = 3.14;
      a= 100;
      S = ‘всем привет’;
      K = $;
```

Описание переменных

Var *имя_переменных: тип_данных;*

```
 var a, b, c: integer;
     X, y: real;
```

Тип переменной определяет:

1. Возможный набор для данных переменных
2. Размер его внутреннего предоставления в памяти компьютера
3. Набор допустимых операций, которыми можно выполнять с переменными данными типами

III. Раздел выполняемых действий (тело программы)

Begin

…

End.

Пример программы

Program Primer; *{заголовок программы}*

{раздел описания}

Const …

Var …

Begin *{начало программы}*

…

End. *{конец программы}*

Типичные ошибки, которые могут возникнуть:

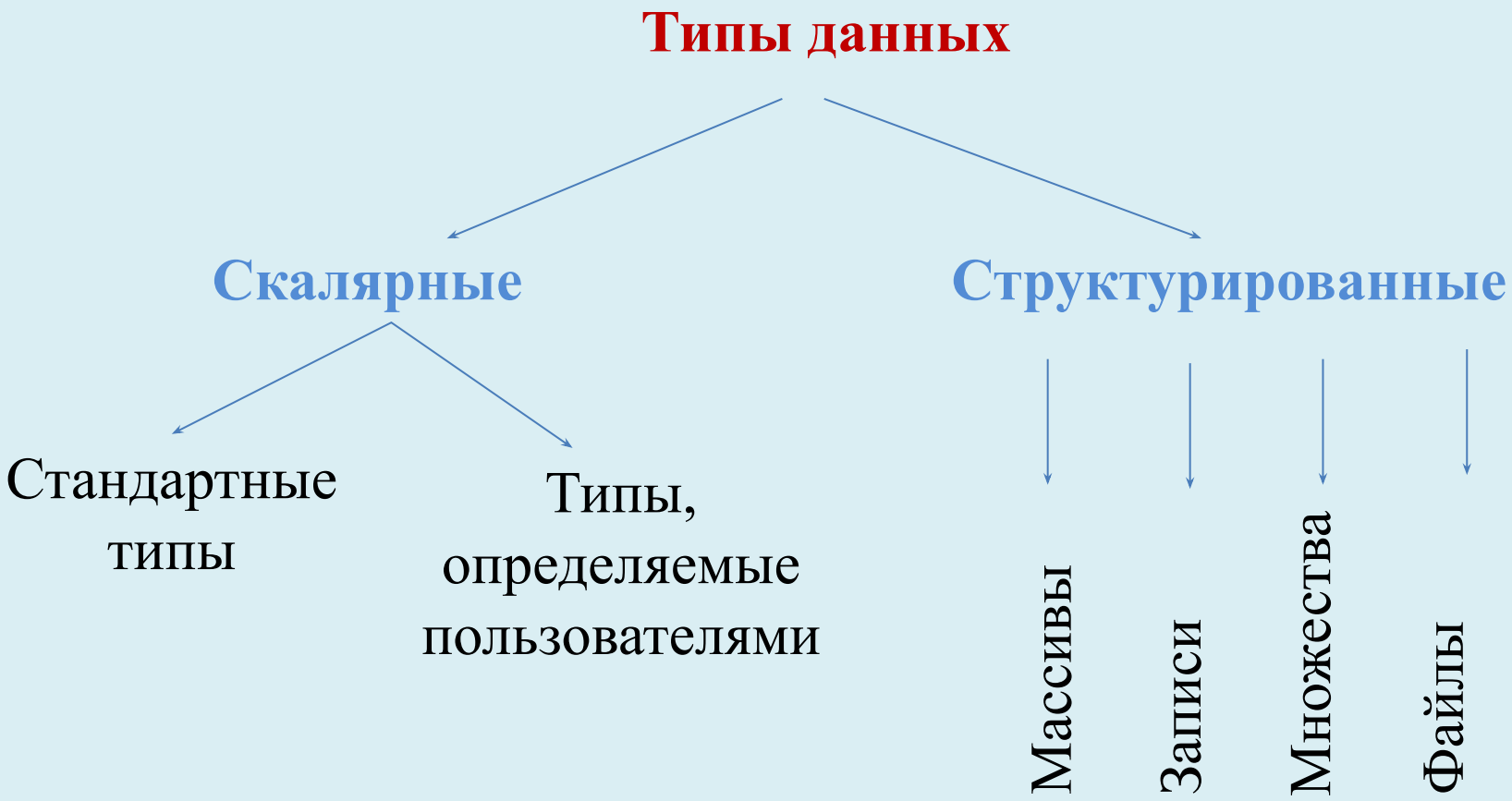
1. Использование в инструкции программирования неописанных переменных.
2. Отсутствует точка с запятой в конце
3. В конце программы отсутствует точка
4. Имена переменных не должны совпадать не с именем программы, не с каким-либо служебным символом.
5. Var a:integer;
   const x=10;

Типы данных в Delphi

Одним из важнейших принципов современного программирования является принцип строгой типизации данных. В соответствии с которыми, каждая используемая в программе переменная должна быть отнесена к определенному типу

Тип данных определяет:

1. Возможные значения переменных const, функций
2. Определяет внутреннюю форму представлений данных в памяти компьютера
3. Операции и функции, которые могут выполняться под величинами, принадлежащие к данному типу


Типы данных
Скалярные
Структурированные
Стандартные типы
Типы, определяемые пользователями
Массивы
Записи
Множества
Файлы

Стандартные типы данных

В Delphi к стандартным относятся 5 типов:

1. *Целый*
2. *Вещественный*
3. *Логический*
4. *Символьный*
5. *Строковый*

Целый тип данных

Переменные целого типа могут иметь разную байтовую длину, могут быть знаковыми или безнаковыми.

Тип	Название	Размер	Знак	Диапазон значений
shortint	Короткое целое	1 байт	Есть	-128..127
smallint	Малое целое	2 байта	Есть	-32768..32767
integer, longint	Длинное целое	4 байта	Есть	-2147483648.. 2147483647
int64	64-разрядное целое	8 байт	Есть	-9223372036854775808.. 9223372036854775807
uint64	64-разрядное целое	8 байт	Нет	0.. 18446744073709551615
byte	Байт	1 байт	Нет	0..255
word	Слово	2 байта	Нет	0..65535
longword	Двойное слово	4 байта	Нет	0..4294967295

Все вычисления и другие преобразования в программе записываются в виде выражений. Выражение включает несколько операций, которые выполняются в порядке приоритетности.

Различают:

- *Арифметические операции*
- *Операции отношения*
- *Логические операции*
- *Строковые операции*
- *Операции над множествами*

Арифметические операции: **+, -, *,** целочисленное деление **div**, взятие остатка от деления **mod**. Результат арифметической операции над целыми типами - есть величина целого типа.

Операции отношения: =, <>, >=, <=, >, <

Результат операции отношения над целыми числами логического типа **TRUE** или **FALSE**

Функции результатом, которых являются целые типы.

ABS (x) = **I**x**I**, **SQR** (X) = x^2

Существуют стандартные функции для целого типа, которые дают вещественный результат: **sin**(x), **cos**(x), **arctan**(x), **ln**(x), **exp**(x), **sqrt**(x)

Описание:

Var I,j: **integer;**

B1, b2, b3: **byte;**

Mylong: **longint;**

Вещественный тип данных

Используется при обработке чисел с дробной частью 3 основных типа:

Real – основной (6 байтов)

Double – с двойной точностью (8 байтов)

Extended – расширенный (10 байтов)

Операции: +, -, *, /

Trune (x) – выделение целой части, отсечением дробной части

Round(x) – аргумент до ближайшего целого числа

X=34,75

Trune(x) = 34

Round(x) = 35

Описание:

```
Var x,y: real;
d: extended;
```

Логический тип данных

Любая логическая переменная может принимать только **одно из двух** возможных значений **true** (истина) или **false** (ложь).

Основные операции:

*Логическое отрицание (**not**);*

*Логическое сложение (**or**);*

*Логическое умножение (**and**).*

Правила выполнения логических операций

Логическое умножение:

(false) **and** (false) = false

(false) **and** (true) = false

(true) **and** (false) = false

(true) **and** (true) = true

Логическое сложение:

(false) **or** (false) = false

(false) **or** (true) = true

(true) **or** (false) = true

(true) **or** (true) = true

Логическое отрицание:

not (true) = false

not (false) = true

Пример:

Дано: *x=10; a=15; b=0; c=30;*

Найти:

(x>0) and (a<10) = ?

(b<15) or (c>100) = ?

(a=0) or (b>-15) or (c>10) and (x=10) =?

Символьный тип данных

В качестве своего значения переменная может иметь один любой символ из заданного набора символов

Описание: используется ключевое слово **char**

Операции:

Каждому символу соответствует свой числовой код в таблице символов

Функция **ord (**символ**)** дает код символа;

Функция **chr (**код_символа**)** дает сам символ.

Пример:

Ord (‘A’) = 65

Chr (65) = ‘A’

Строковой тип

Переменная строкового типа в качестве своего значения может иметь любую текстовую строку длиной **не более 255** символов.

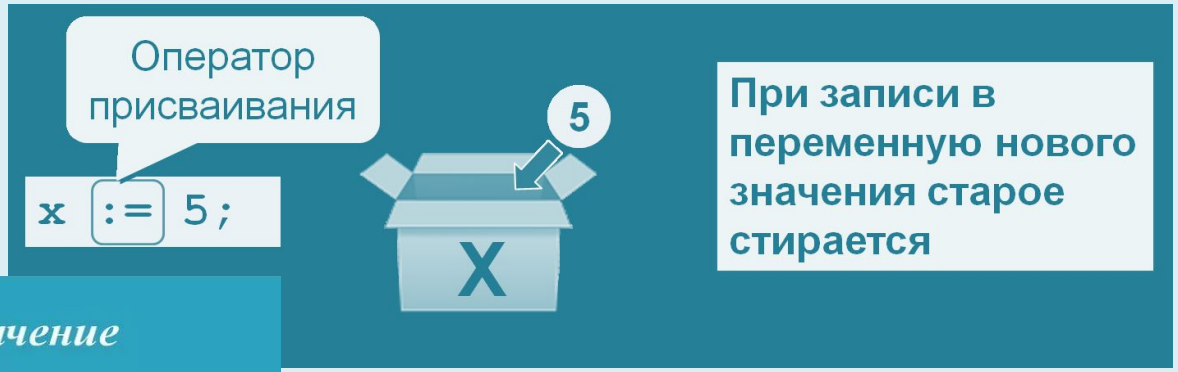
Для описания используется ключевое слово **string**

```
Var stroka: string;
    name: string[20];
```

Пример:

```
Var s1:string;
s2: string[3];
Begin
S1:=‘Всем привет’;
S2:= ‘хорошо’;
End;
```

Тема 4. **Основные инструкции в Delphi (Паскаль)**



Запись на Pascal	*Выражение*	*Значение*
:=	x := 5	Заменяет значение переменной
+=	x += n	Увеличивает число на n
-=	x -= n	Уменьшает число на n
*=	x *= n	Увеличивает число в n раз
/=	x /= n	Уменьшает число в n раз

Инструкция (оператор) – это специальная совокупность служебных слов, индификаторов и специальных знаков, выполняющих определенные действия.

В каждой инструкции ставятся: ‘;’

Инструкция присваивания

Инструкция позволяет установить переменной какое-либо значение, возможно выполнить при этом вычисление:

Переменная:=выражение;

При использовании инструкции присваивания необходимо выполнять правила:

1. *Все переменные в левой и правой части инструкции присваивания должны быть отнесены к соответствующему типу;*
2. *Все переменные в правом выражении должны иметь конкретное значение;*
3. *Типы правой и левой части должны совпадать*

Инструкция ввода

Read(список переменных);

или **Readln**(список переменных);

Например:

```
Var i, j, k: integer;
      x,y: real;
      st: string;
…
Read (i,j,k);
Read (x,y);
Readln (st);
```

Инструкция ввода

В Delphi используется служебное слово **Write ()** или **Writeln(**список переменных**)**

При использовании инструкции **write(**st**)** после вывода курсор остается в той же строке за последним выводом символа.

При использовании **Writeln(**st**)** после вывода данных курсор устанавливается в 1-ой позиции в следующей строке

Например:

Необходимо вывести значения переменных a, b, c в 3-х строках экрана

```
Writeln (a);
Writeln (b);
Writeln (c);
```

Оператор вывода **Writeln** без параметров, он просто переводит курсор на начало новой строки

Для одновременного вывода нескольких переменных можно использовать

Write (a, b, c**)**;

A – целочисленное

B – вещественное

C – текстовое

Оператор вывода позволяет кроме переменных *выводить на экран текст*. Текст для вывода необходимо записывать в апостроф

Write (‘first program’**)**;

При выводе в самом операторе можно производить элементарные арифметические действия

Write (10*15*2**)**;

Например:

```
Writeln (54*a+sin(a));
B:= 54*a+sin(a));
Writeln (b);
```

Формат вывода для числовых элементов

Количество позиций, которое будет отводиться для вывода значения переменной.

Например:

Write (i:3);

Для задания формата ставится ‘:’ и выделяется позиция.

Формат вывода для вещественных чисел

Write (x:7:2)

7 – количество позиции на все вещественное число, включая точку и знак

3 – количество знаков после запятой

Например: x=123,4536

Write (x:9:4);

В инструкции вывода можно комбинировать вывод значений переменных и текста

Например:

Write (‘number:’, i:2, ‘x=‘, x:5:2, ‘y=‘, y:9:3);

При одновременном использовании инструкции ввода и вывода в программе можно организовать простейший диалог с пользователем.

Пример:

```
Write (‘input x=‘);
Readln (x);
Write (‘input y=‘);
Readln (y);
```

Пример 1:

Найти сумму 2-х введенных с клавиатуры целых чисел, результат вывести на экран.

```
Program Pr1;
Var sum, a, b: integer;
Begin
Writeln (‘введите первое число’);
Readln (a);
Writeln (‘введите второе число’);
Readln (b);
Sum:= a+b;
Writeln (sum);
Readln;
end.
```

Пример 2:

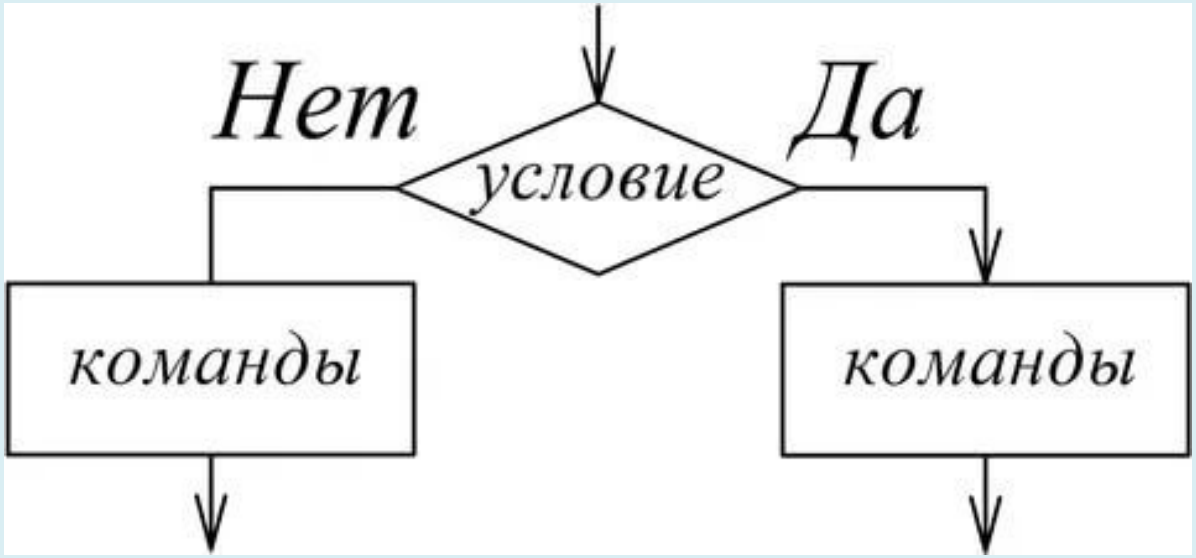
Найти площадь треугольника по трем его сторонам, используя следующую форму:

P:=(a+b+c)/2;

S:=$\sqrt{p*(p-a)*(p-b)*(p-c)}$

```
Program Pr2;
Var c, a, b: integer; p,s:real;
Begin
Writeln (‘введите сторону а’);
Readln (a);
Writeln (‘введите сторону в’);
Readln (b);
Writeln (‘введите сторону с’);
Readln (c);
P:=(a+b+c)/2;
S:=SQRT(p*(p-a)*(p-b)*(p-c));
Writeln (‘площадь =‘, s:8:3);
Readln;
end.
```

Тема 5. **Управляющие операторы языка**



Для организации разветвленной и циклической структур используются управляющие операторы языка, которые определяют последовательность выполнения действий в программе.

Оператор условной передачи управления

Инструкция проверки условий

Существуют две разновидности проверки условий:

1. Простая – осуществляется проверка с выбором одной из двух возможных альтернативных действий
2. Множественная – с возможностью перехода на одну из нескольких альтернатив.

Общий вид простой инструкции проверки условий:

If *условие* **then** *инструкции* **else** *набор инструкций*;

В условие записывается логическое выражение, результат которого может быть либо **true**, либо false.

Истинность условия приводит выполнение инструкций к ветки **then**, *ложность* к выполнению ветки **else.**

If *a>b* ***then*** *x:=a+b* ***else*** *x:=a-b;*

В простейшем случае инструкция включает в себя 1 инструкцию .

Если в ветки **then** или **else** необходимо выполнить больше , чем 1 инструкцию, то их необходимо заключить в операторные скобки **begin … end.**

Пример:

```
If x>0 then begin
    x:=x+8;
    Writeln (x);
    end
    else begin
    Y:=y-2;
    Writeln (y);
    end.
```

В частном случае может отсутствовать ветка **else**.

If *условие* ***then*** *набор инструкций;*

Пример:

If x>0 **then** x:=x+1;

Вложенные условные инструкции

If *условие1* ***then*** *блок инструкций1* ***else***
if *условие2* ***then*** *инструкция2* ***else*** *инструкция3;*

Пример1:

Среди двух введенных с клавиатуры чисел найти максимальное.

```
Program Pr1;
Var a, b: integer;
Begin
    Writeln (‘введите 1 число’);
    Readln (a);
    Writeln (‘введите 2 число’);
    Readln (b);
    if a>b then Writeln (‘Максимальное число =‘,a)
    else Writeln (‘Максимальное число =‘,b);
Readln;
End.
```

Пример 2:

Вычислите значение функции $y = \begin{cases} x * \cos(2 * x), x < 5 \\ \frac{8*x+12}{x}, x \geq 5 \end{cases}$

```
Program Pr2;
Var x,y: real;
Begin
Writeln (‘введите x’);
Readln (x);
If x< 5 then  y:=x*cos(2*x)
          else y:=(8*x+12)/x;
Writeln (‘y =‘, y:6:4);
Readln;
End.
```

Пример 3:

Вычисление корней из квадратного уравнения

```
Program Pr3;
Var a,b,c,d,x1,x2: real;
Begin
Writeln (‘введите a’);
Readln (a);
Writeln (‘введите b’);
Readln (b);
Writeln (‘введите c’);
Readln (c);
D:= b*b – 4*c*a;
If d<0 then Writeln (‘no’)
        else if d=0 then
        begin
             x1:=-b/(2*a);
             Writeln (‘Only one:’, x1:7:2);
        end
            else
                begin
              x1:=(-b+SQRT(d)/(2*a);
                      x2:=(-b-SQRT(d)/(2*a);
              Writeln (‘x1=‘, x1:7:2, ‘x2=‘, x2:7:2);
        end;
Readln;
end.
```

Множественная инструкция проверки условия (инструкция выбора)

Данная инструкция позволяет выбрать одно из нескольких действий. Работа инструкции основана на использовании специальных переменных и называются **селектором.**

Общий вид:

Case *селектор* ***of***
Селектор1: набор_инструкции1;
Селектор2: набор_инструкции2;
Селектор3: набор_инструкции3;
…….
СелекторN: набор_инструкцииN;
Else *другой_набор_инструкции1;*
End;

Селектор – переменная, либо выражение, значение которого может быть распределено перед выполнением инструкции выбора.

В качестве селектора мы можем использовать переменные только ***целого типа*** или ***символьного типа.***

В качестве *значения селектора* в теле инструкции можно указывать:

- одиночные допустимые значения селектора;
- списки допустимых значений;
- интервалы допустимых значений.

Одиночные допустимые значения:

```
Var i: integer;
Case I of
        1: инструкция1;
        2: инструкция2;
        3: инструкция3;
Else инструкция значений селектора
End;
```

Списки допустимых значений

```
Var an: integer;
Case an of
        1, 2 ,10: инструкция1;
        3,5,7,11: инструкция2;
        4,6: инструкция3;
End;
```

Интервалы допустимых значений

```
Var  s: char;
Case s of
        ‘a’..’z’: инструкция1;
        ‘A’..’Z’: инструкция2;
        ‘0’..’9’: инструкция3;
End;
```

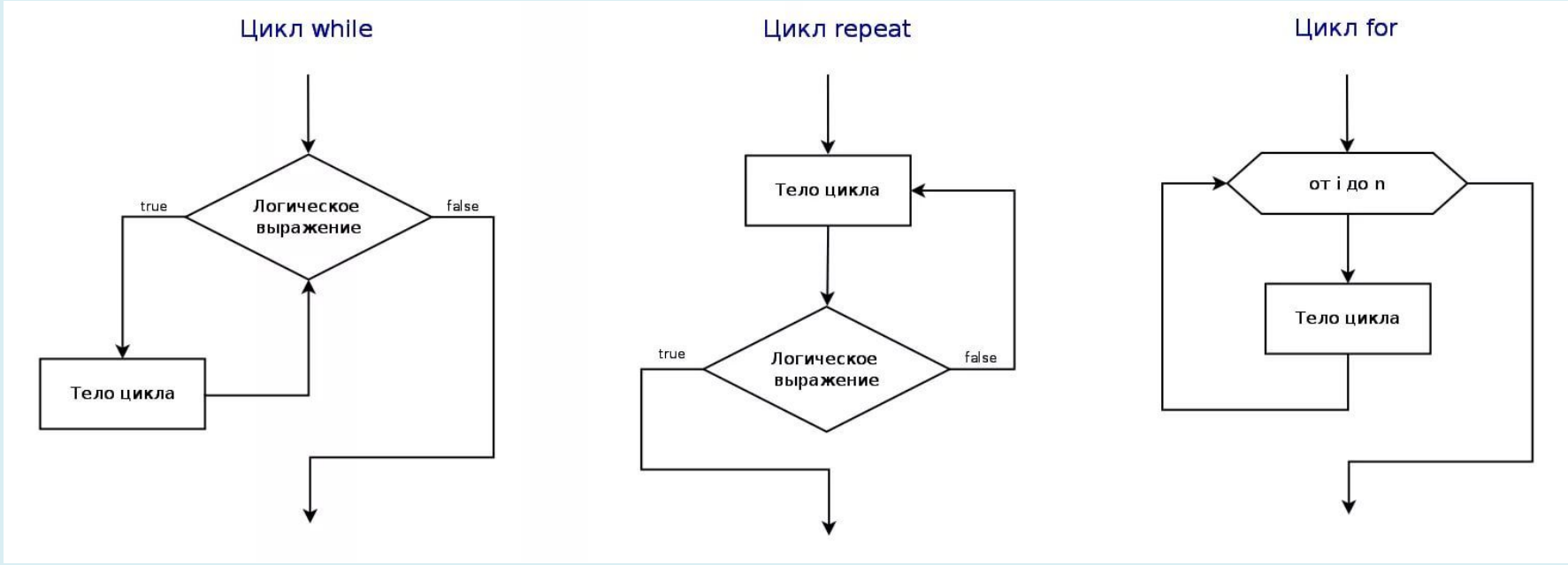
Пример1:

Перевести введенный с клавиатуры пользователем номер дня недели в его словесную форму, используя инструкцию выбора.

```
Program Pr1;
Var nomer:byte;
Begin
    Writeln (‘введите день недели’);
    Readln (nomer);
Case Nomer of
1: Writeln (‘Понедельник’);
2: Writeln (‘Вторник’);
3: Writeln (‘Среда’);
4: Writeln (‘Четверг’);
5: Writeln (‘Пятница’);
6: Writeln (‘суббота’);
7: Writeln (‘воскресенье’);
Else Writeln (‘неправильный номер’);
End;
Readln;
end.
```

Если в инструкции выбора для каких-либо значений необходимо выполнить больше чем одну инструкцию, в этом случае ставятся для данного значения операторные скобки **begin …end.**

Тема 6. **Управляющие операторы языка программирования. Инструкции циклов**



Инструкции циклов позволяют **многократно** в программе выполнять набор одних и тех же действий с разными данными.

Существуют две разновидности циклов:

- Циклы с **известным** числом повторений
- Циклы с **неизвестным** числом повторений

Циклы с известным числом повторений

В основе работы цикла лежит использование специальной переменной, которая называется **счетчиком (параметром) цикла**.

Для счетчика задается *начальное* и *конечное значение*. При каждом повторение цикла, значение счетчика увеличивается **на 1** и сравнивается с заданным конечным значением. Работа цикла прекращается, когда текущее значение счетчика цикла на 1 превысит конечное значение.

Общий вид:

For *счетчик_цикла:=начальное_значение* ***to*** *конечное_значение* ***do*** *действие;*

Пример:

For k:=1 **to** 3 **do** Write (‘Привет’);

1) k=1 1>3 – Привет
2) k=2 2>3 – Привет
3) k=3 3>3 – Привет
4) k=4 4>3 – {Завершение цикла}

В качестве счетчика цикла могут использоваться переменные только *целого* или *символьного типа*.

Начальное значение счетчика цикла не может быть больше конечного значения счетчика цикла.

Существует разновидность, когда счетчик цикла наоборот уменьшается **с началом -1**

***For** k:=10 **downto** 1 **do**…*

Пример 1.

Вычислить сумму первых десяти натуральных чисел.

```
Program Pr1;
Var sum, k: integer;
Begin
    sum:=0;
    for k:=1 to 10 do
        sum:=sum + k;
    Writeln (‘сумма =‘, sum);
end.
```

Пример 2.

Вывести на экран в обратном порядке 51 число (от 50 до 0).

```
Program Pr2;
Var i: integer;
Begin
    for i:=50 downto 0 do
    Write (i:3);
end.
```

Пример 3.

Вывести на экран в прямом порядке заглавные буквы латинского алфавита, в обратном порядке строчные буквы.

```
Program Pr3;
Var k: ichar;
Begin
    for k:='A' to 'Z' do
    Write (k:3);
Writeln;
    for k:='z' downto 'a' do
    Write (k:3);
end.
```

Если в теле цикла по условию задачи необходимо выполнить *больше чем одну* инструкцию, то ставятся операторные скобки **begin…end**.

Пример 4.

Вывести на экран 100 натуральных чисел в прямом порядке, затем возвести в квадрат эти числа и вывести.

```
Program Pr4;
Var p, k: integer;
Begin
    for k:=1 to 100 do
        Write (k:3);
    Writeln (‘квадраты чисел’);
    for k:=1 to 100 do
        begin
            p:=sqr(k);
            Write (k:3);
        End;
end.
```

Существуют вложенные циклы, когда в теле цикла в качестве инструкции запускается второй цикл. В этом случае для каждого цикла используется свой счетчик цикла.

For k:=1 **to** 5 **do**

for i:=1 **to** 3 **do** действия;

1) *k=1*

i=1 действие;

i=2 действие;

i=3 действие;

2) *k=2*

i=1 действие;

i=2 действие;

i=3 действие;

3) …..

Пример 5.

Вывести два числа, найти их сумму, возвести сумму в степень введенную с клавиатуры.

```
Program Pr5;
Var m, a, b, s, n, k: integer;
Begin
    Writeln (‘Введите два числа’);
    Readln (a,b);
    S:=a+b;
    Writeln (‘введите степень’);
    Readln (n);
    M:=1;
    for k:=1 to n do
        m:=s*m;
        Writeln (‘сумма =’, s, ‘сумма в степени =‘);
    end.
```

Цикл с неизвестным числом повторений

Число повторений в данных циклах определяется заданным логическим условием.

Существуют две разновидности циклов:

- Цикл с предусловием (сначала проверка условия, если оно истинно, выполняется действие)
- Цикл с постусловием (сначала выполняется действие, затем проверяется условие)

Цикл с предусловием

Общий вид:

While *условие* ***do*** *действие;*

Данный цикл работает, пока условие истинно, как только условие ложно, цикл завершает работу.

Пример:

```
X:=10;
While x>10 do begin
                Writeln (x);
                X:=x-2;
                End;
```

1) *10>0 true x=8*
2) *8>0 true x=6*
3) *6>0 true x=4*
4) *4>0 true x=2*
5) *2>0 true x=0*
6) *0>0 false*

Цикл с предусловием **While** может не выполниться ни разу, т.к. условие может быть ложным в самом начале, что является признаком окончания цикла.

Переменная, которая участвует в проверке условия, до цикла должна быть определена.

Пример:

```
X:=10;
While 10>0 do Writeln (x);
```

Происходит зацикливание, т.е цикл выполняется бесконечное число раз, т.к. x неизменна и условие постоянно истинно.

Переменная, которая участвует в проверке условия в теле цикла, **обязательно должна изменять свое значение**, чтобы на каком-то очередном шаге условие оказалось ложным и цикл завершил свою работу.

Пример 6:

Найти наибольший общий делитель.

```
Program Pr6;
Var a, b: integer;
Begin
    Writeln (‘введите два числа’);
    Readln (a,b);
    While a<>b do
        if a>b then a:=a-b
            else a:=b-a;
    Writeln (‘наибольший общий делитель’, a);
End;
```

Цикл с постусловием

Общий вид:

Repeat *действие1;*
....
действие n;
Until *условие;*

Цикл выполняется при *ложном* условии, как становится истинным он завершает свою работу.

Повторяем действие до тех пор, пока условие не станет истинным.

Пример:

```
X:=10;
Repeat
    Writeln (x);
     X:=x-2;
Until x=o;
```

1) 10 x=8 8<>0 false
2) 8 x=6 6<>0 false
3) 6 x=4 4<>0 false
4) 4 x=2 2<>0 false
5) 2 x=0 0<>0 true

Цикл **Repeat** всегда выполняется хотя бы 1 раз, т.к. условие проверяется в конце после выполнения действия.

Пример 7

Применение цикла **Repeat** для реализации многократного повтора одной и той же программы.

```
Program Pr7;
Var a, b: integer; otv: char;
Begin
repeat
Writeln (‘введите два числа”);
Readln (a,b);
    If a>b Writeln (‘max=‘,a)
                Else writeln (‘max=,b);
Writeln (‘хотите продолжить y/n’);
Readln (otv);
Until onv = ‘n’;
en.d.
```

Пример 8.

Среди введенных пользователем с клавиатуры положительных чисел найти максимальное число.

REPEAT …UNTIL

```
Program Pr8_1;
Var k, max: integer;
Begin
Writeln (‘введите число’);
Readln (k);
Max:=k;
   While k>0  do begin
          If k>max then max:=k;
Writeln (‘введите число’);
Readln (k);
End;
          If max<0 then Writeln (‘нет’)
         else Writeln (‘max=‘, max);
end.
```

WHILE ...DO

```
Program Pr8_2;
```

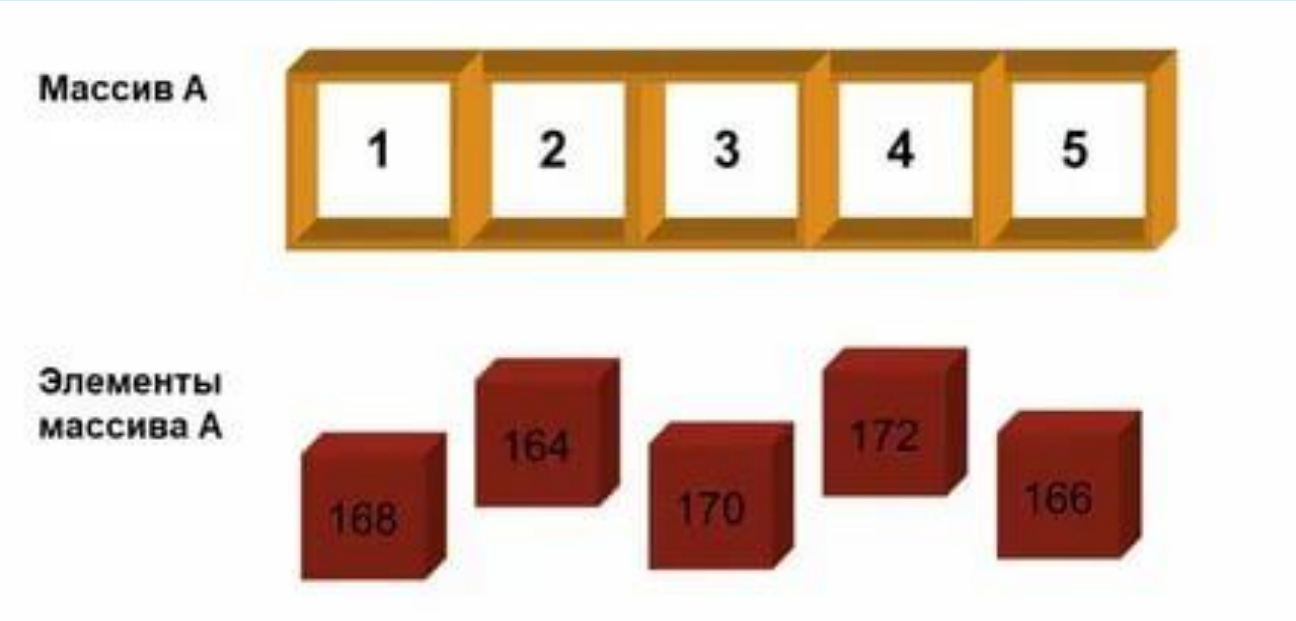
Цикл While удобно использовать для проверки корректности введенного пользователем данных. В особенных случаях, когда с неточными данными может возникнуть программная ошибка.

Например:

Необходимо, чтобы введенное число было положительным

```
Begin
Writeln ( ‘введите положительное число’);
Readln (k);
   While k<0 do begin
Writeln (‘ввели неправильное значение’);
Writeln (‘введите положительное число’);
Readln (k);
End;
```

Тема 7. **Массивы**



Массив – это структура данных, содержащая ограниченное число однотипных элементов.

В качестве элементов массива могут быть элементы **любого** из существующих типов данных.

Типы данных:

- целые;
- вещественные;
- логические;
- символьные.

Например: 1 2 3 4 5.

Каждый элемент хранится **последовательно** друг за другом, линейная структура.

Для описания программы необходимо задать **имя**, **количество** элементов в массиве (*начальным и конечным*), **тип** элементов массива.

Общий вид описания массива:

Имя_переменной: ***array [****начало конец****] of*** *тип_элементов;*

Например:

```
Var mas1: array [1..100]  of integer;
Var mas2: array [0..200]  of char;
Var mas3: array [-10..10]  of real;
```

При разработке массива **типичными задачами являются**:

- Поиск в массиве наименьшего, наибольшего, среднего значения;
- Поиск первого, последнего или всех появлений заданного элемента;
- Сортировка массива, т.е. выстраивание элементов массива в соответствии с заданным порядком.

Массивы мы можем использовать только для **временного** хранения информации.

При этом при запуске программы, для каждой переменной должна выделяться **своя область памяти**.

Размер массива – число элементов и тип массива.

Общий объем определяется, как произведение числа на байт типа.

*100*4 = 400 байт – для массива целых чисел*
*201*1 = 201 байт – для массива символьного типа*
*21*6 = 126 байт – для массива вещественного типа*

В массиве элементы хранятся друг за другом.

Например:

Номер элемента	1	2	…	10		100
Эл-т массива	-99	0	…	13		77

Для любого массива для каждого элемента, мы можем определить **адрес его размещения в памяти**.

Для этого необходимо:

- Начальный адрес массива
- Порядковый номер элемента
- Байтовый размер элемента

Тогда адрес будет выглядеть по формуле:

*Адрес = нач_адрес + (номер_элемента – 1)*размер_элемента*

Работа с отдельными элементами массива

Допустимые операции: *все те операции*, которые разрешены для базового типа элементов массива

Каждый элемент массива при *обращение* к нему задается: ***имя_переменной[номер_элемента]***

Например: mas[3] – третий элемент массива

Порядковый номер чаще всего задается при помощи переменной целого типа.

Индексная переменная должна быть объявлена отдельно.

Например:

```
Var i:=5;
Mas[i]:=45;
```

В качестве индексной переменной можно использовать индексное выражение при обращение к элементу, при условии, что переменная в выражении на момент обращения известна и результат вычисления целый тип.

```
Const n=10;
Var b: array [1..100] of integer;
K:=integer;
…
Readln (k);
B[2*k*n]:=15;
…
```

С каждым элементом массива можно выполнять тот набор действий, который разрешен для соответствующего типа элементов, из которых состоит массив.

При обработке массивов чаще всего используются циклы с известным числом повторения.

Пример 1.

Создать из 10 целых чисел путем ввода с клавиатуры, вывести в прямом и обратном порядке.

```
Program pr1;
Var masINT: array [1..10] of integer;
i: integer;
Begin
For i:=1 to 10 do
Readln (mas[i]);
Writeln (‘прямой порядок’);
For i:=1 to 10 do
Write (mas[i]:3);
Writeln;
Writeln(‘в обратном порядке’);
For i:=10 downto 1 do
Write (mas[i]:3);
end.
```

Пример 2

Создать из 10 целых чисел и вывести на экран квадраты этих чисел.

```
Program pr2;
Const n=10;
Var mas: array [1..n] of integer;
Rez,i: integer;
Begin
    For i:=1 to n do
        Readln (mas[i]);
            For i:=1 to n do begin
                Rez:=mas[i]*mas[i];
                Write (mas[i]:5, ‘квадрат числа’, rez:7);
                End;
end.
```

Пример 3

Создать массив из 100 случайных чисел в диапазоне от 0 до 99.

```
Program pr3;
Const n=100;
Var mas: array [1..n] of integer;
i: integer;
Begin
Randomize;
    For i:=1 to n do
        mas[i]:=Random (100);
            For i:=1 to n do begin
                Write (mas[i]:5);
                 End;
end.
```

-100 до 100

Random (201**)**-100;

-50 до 50

Random (101**)**-50;

Пример 4

Сгенерировать массив из 50 целых чисел в диапазоне от -100 до 100. Вывести на экран элементы массива и сумму всех элементов.

```
Program pr4;
Const n=50;
Var mas: array [1..n] of integer;
Sum, i: integer;
Begin
Randomize;
    For i:=1 to n do
        mas[i]:=Random (201) - 100;
 Sum:=0;
            For i:=1 to n do begin
                Write (mas[i]:5);
                 Sum:=sum+mas[i];
                 End;
Writeln (‘сумма равна =‘, sum);
Readln;
end.
```

Пример 5

Сгенерировать массив из 100 целых чисел в диапазоне от 0 до 50. Найти в массиве все появления заданного числа, введенного пользователем с клавиатуры, с выводом на экран позиции этого элемента в массиве и подсчетом количества появлений заданного числа.

```
Program pr5;
Const n=100;
Var mas: array [1..n] of integer;
kol, I, k: integer;
Begin
Randomize;
    For i:=1 to n do begin
          mas[i]:=Random (51) ;
          Write (mas[i]:5);
          End;
Writeln (‘введите искомое число’);
Readln (k);
Kol:=0;
     For i:=1 to n do
          If mas[i] = k then begin
               Writeln (‘место’, i);
                Kol:=kol+1;
                End;
If kol = 0 then writeln (‘числа нет’)
      else writeln (‘число встречается’, kol, ‘ раз’);
Readln;
end.
```

Пример 6

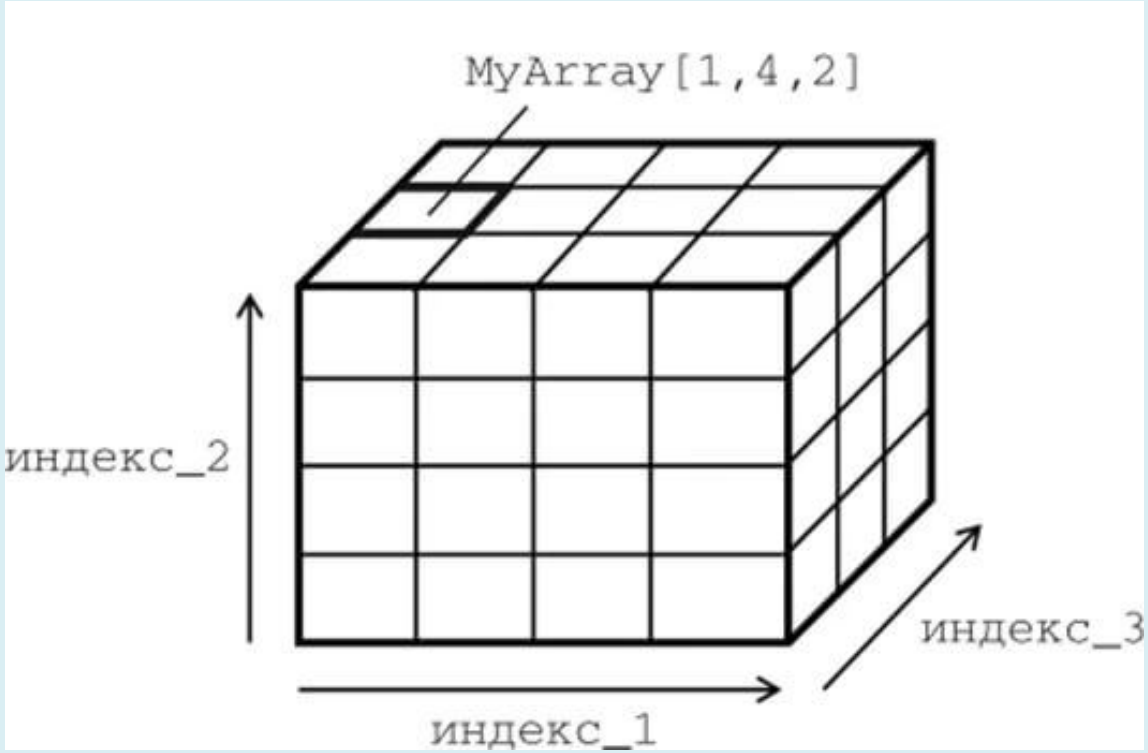
Оформление пользовательского меню с использованием инструкции case и реализация возможности многократного повтора программы.

Сгенерировать массив из 50 целых чисел в диапазоне от -100 до 100 и вывести на экран. Найти в данном массиве:

1. Количество отрицательных элементов
2. Сумму нечетных элементов
3. Вывести на экран элементы, стоящие на нечетном месте массива. Каждое действие оформить как пользовательское меню.

```
Program Pr6;
Const n=50;
Var a:array [1..n] of integer;
I, sum, kol, k: integer;

Begin
Repeat
Randomize;
Writeln (‘1 – генерация и вывод’);
Writeln (‘2 – количество отрицательных элементов’);
Writeln (‘3 – сумма нечетных элементов’);
Writeln (‘4 – вывод элементов на нечетных местах ’);
Writeln (‘5 – выход’);
Writeln;

Readln (k);
Case k of
1: for i:=1 to n do begin
A[I]:=random (201) -100;
Write (a[i]:5);
End;




2: begin
Kol:=0;
For i:=1 to n do
```

Тема 8. **Многомерные массивы**



- **Многомерный массив или массив массивов** - это линейная структура, содержащая ограниченное число однотипных элементов и чаще всего использующаяся для обработки матриц.

В двумерных массивах каждый элемент имеет два индекса: **номер строки**, в которой он находится и **номер столбца**.

Осуществляется прямой доступ каждому элементу по его индексу.

$$\begin{matrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{matrix}$$

Var a: **array** [1..3, 1..4] **of** integer;

В памяти элемента двумерного массива хранятся *последовательно строка за строкой*. При описание двумерного массива мы указываем:

- Имя переменной массива
- Размерность, т.е. количество строк и столбцов
- Тип хранящихся элементов

Общие число массива $n \times m$

Массив[строка, столбец]

Для обработки в двумерных массивах используются *циклы с известным числом повторения*, где внешний цикл будет изменяться по количеству строк, а внутренний по количеству столбцов.

Пример 1

Создать двумерный массив размерностью 3 X 5, путем ввода целого значения с клавиатуры, затем вывести полученный массив в виде таблицы на экран.

```
Program Pr1;
Const n=3;
Const m=5;
Var a: array [1..n, 1..m] of integer;
I,j : integer;
Begin
Writeln (‘введите элементы массива’);
    For i:=1 to n do
          For j:=1 to m do
          Readln (a[I,j]);
     For i:=1 to n do begin
           For j:=1 to m do
           Writeln (a[I,j]:3);
     Writeln;
     End;
end.
```

Пример 2

Сформировать массив 10 X 15 из случайных чисел в диапазоне -100 до 100 и вывести его в виде матрицы на экран.

```
Program Pr2;
Const n=10;
Const m=15;
Var a: array [1..n, 1..m] of integer;
I,j : integer;
Begin
Randomize;
Writeln (‘введите элементы массива’);
    For i:=1 to n do begin
          For j:=1 to m do begin
          a[I,j]:=random (201)-100;
          Writeln (a[I,j]:5);
           End;
    Writeln;
    End;
end.
```

С элементами двумерного массива обычно выполняются те же действия, что и с элементами одномерного массива:

- поиск минимального и максимального элемента;
- нахождение всех появлений или 1ого появления элемента в массиве;
- сортировка элементов по возрастанию или убыванию
- нахождение суммы всех элементов, среднее значение элементов.

Пример 3

Создать массив 4X5 из случайных чисел от -50 до 50. Вывести его на экран в виде матрицы. Найти в данном массиве минимальное и максимальное значение с выводом их координат на экран.

Оформить каждые действия при помощи пользовательского меню.

```
Program Pr6;
Const n=4;
Const m=5;
Var a:array [1..n, 1..m] of integer;
I, j, min, max, imin, imax, jmin, jmax, k: integer;
Begin
Randomize;
Repeat
Writeln (‘1 – генерация и вывод’);
Writeln (‘2 – поиск максимального значения’);
Writeln (‘3 -  поиск минимального значения’);
Writeln (‘4 – выход’);
Readln (k);
Case k of
1: for i:=1 to n do begin
        For j:=1 to m do begin
        A[I,j]:=random (101) -50;
        Write (a[I,j]:5);
         End;
   Writeln;
   End;
2: begin
Max:=a[1,1];
Imax:=1;
Jmax:=1;
```

Так как положение любого элемента в двумерном массиве можно определить по его индексам: **номера строки** и **номера столбца**, то и адрес нахождения элемента в памяти компьютера можно рассчитать по формуле:

*Адрес_эл-та = нач_адрес_массива + ((номер_строки – 1) * (число_эл-тов_в_строке) + (номер_столбца – 1)) * размер_эл-та*

Пример:

Массив 10 х 15 целых чисел. Необходимо найти адрес пятого элемента в 3-ей строке.

((3-1)*15 + (5-1))*4 = 136 байтов

A: **array** [1..10, 1..5, 1..15] **of** integer;

10 * 5 * 15 = 705 элементов

Type
Имя_типа = тип_описания;
Var *тип_переменная: имя_типа;*

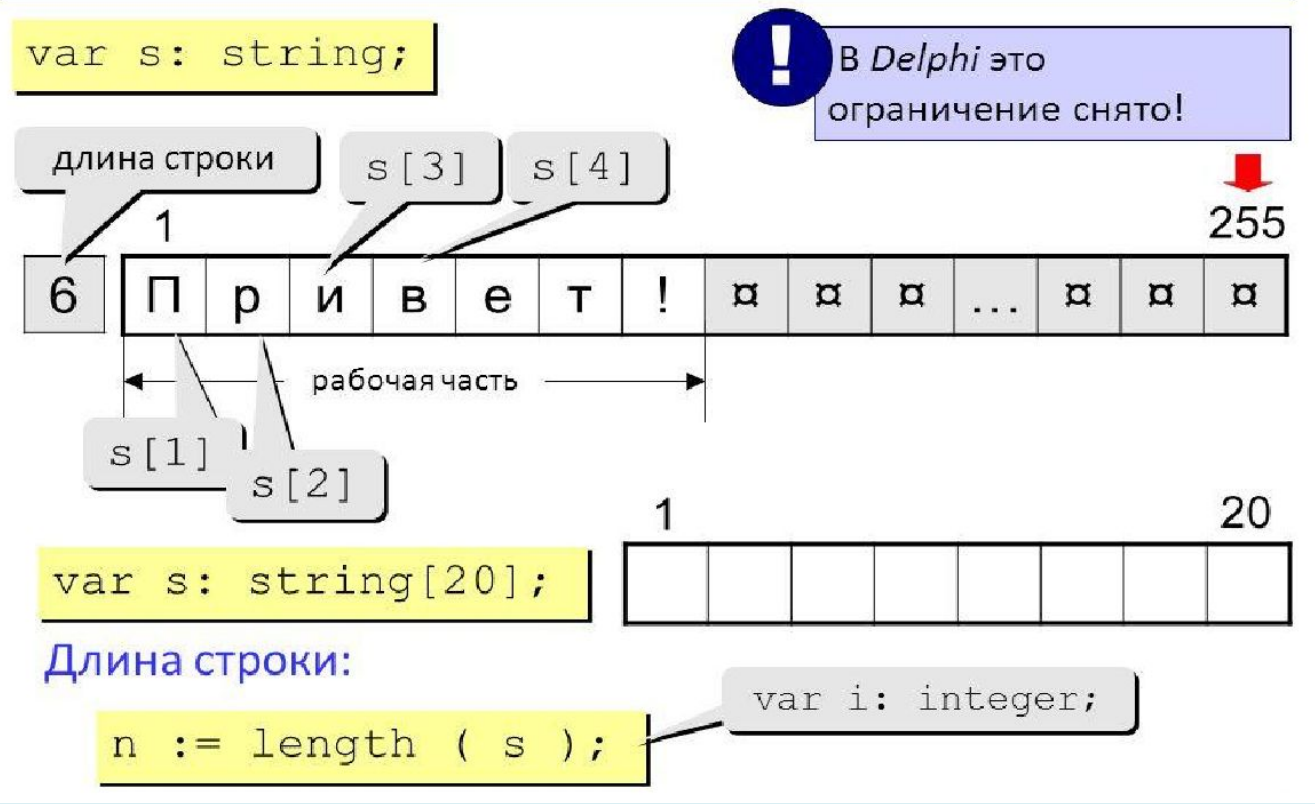
Пример 1:

```
Program Mas1;
Type k: integer;
Var I, n: k;
```

Пример 2:

```
Type
 Tmas = array [1..30] of integer;
TMasChar = array [1..20] of char;
Var
Mas1: array [1..30] of Tmas;
MChar: TMasChar;
Mas2: Tmas;
MChar1: array [1..100] of TMasChar;
```

Тема 9. **Базовая структура данных в символьной строке**



```
Var
St1: string; {максимальная длина = 255 символов}
st2: string[50]; {длина = 50 символов}
St3: string[5];

St1:='Hello';
```

При помощи стандартной функции **Length(строка)** находится длина заданной строки.

Над строками могут выполняться действия:

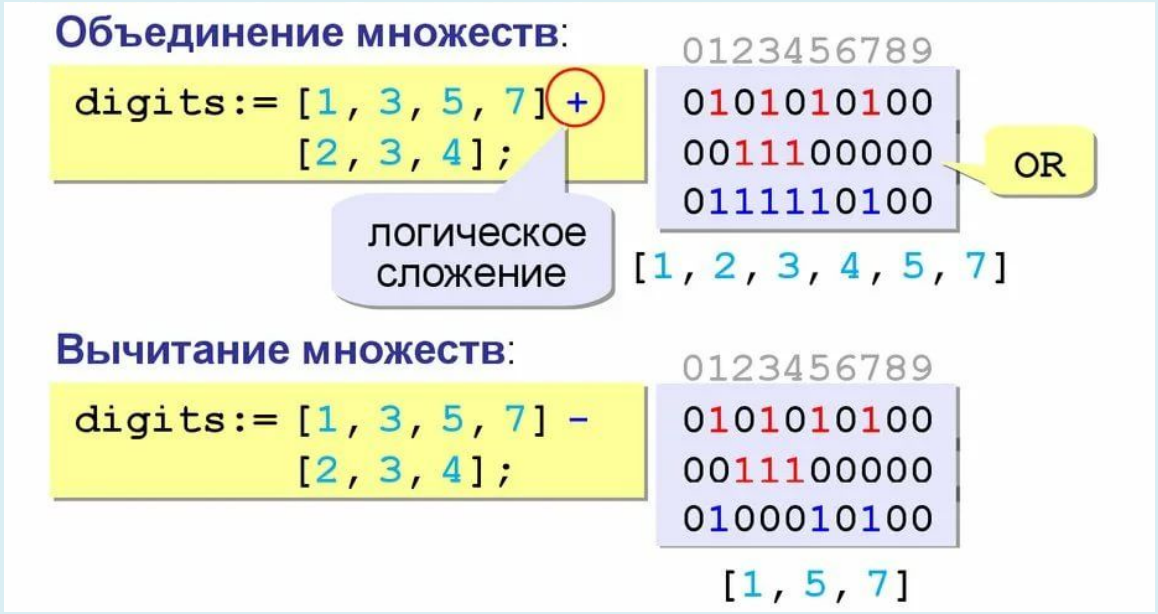
- *Доступ символов к строке*
  st[5]
- *Присваивание строк*
  st:='1';
- *Конкатенация строк (объединение)*
  A:='abcd' + 'A';
  A:=st1 + 'abcd' + st2;
- *Операция отношения (=, <, >, <>)*
  'Hello' < 'hello'
  'He llo' < 'Hel lo'
- *Ввод и вывод строки*
  Readln (st);
  Writeln (st);

Пример 1

Вывести в прямом и обратном порядке текстовую строку длиной не более 80 символов.

```
Program P1;
Var st: string[80];
I,n: byte;
Begin
Readln (st);
N:=length (st);
    For i:=n downto 1 do
    Write (st[i]);
Writeln;
End.
```

Тема 10. **Структуры данных типа множества**



Множество – это структура, которая позволяет объединять некоторое количество однотипных элементов.

Количество элементов множеств может быть переменным, максимально не может быть <256. Порядок следования элементов множеств значений не имеет.

Общий вид:

***Type** имя_типа **= set of** базовый_тип;*

```
Type
   T1 = set of ‘a’..’z’;
   T2 = set of ‘0’..’9’;
   T3 = set of 0..50;
Var
   M1, m2, m3: T1;
…
m1:=[ ] {m1- пустое множество}
M2:= [‘s’];
M3:=[‘a’, ‘b’, ‘c’, ‘d’];

Var s1, s2: T3;
…
S1:=[1,3,5,7,12,15];
s1:=S2;
S1:=[3];
```

Операции над множествами

S2:=[a];

S1:=[a,b,c,d];

1. *Объединение множеств (+)*

   Результирующее множество содержит не повторяющие элементы не повторяющихся множеств.

   S3:=[a,b,c,d];

2. *Пересечение содержит одинаковое множество (*)*

   S3:=[a];

3. Разность – в результате получается элементы, которых нет во втором множестве.

   S3:=[b,c,d];

4. Проверка включения множества – обозначается парой символов <=, используется в качестве условий в циклах и условных инструкциях.

   If *s2<= s1* ***then*** ....

5. Принадлежность множеств (проверка принадлежности к данному множеству, обозначается **in**

   If *'a' **in** M3* ***then*** ...

Пример1

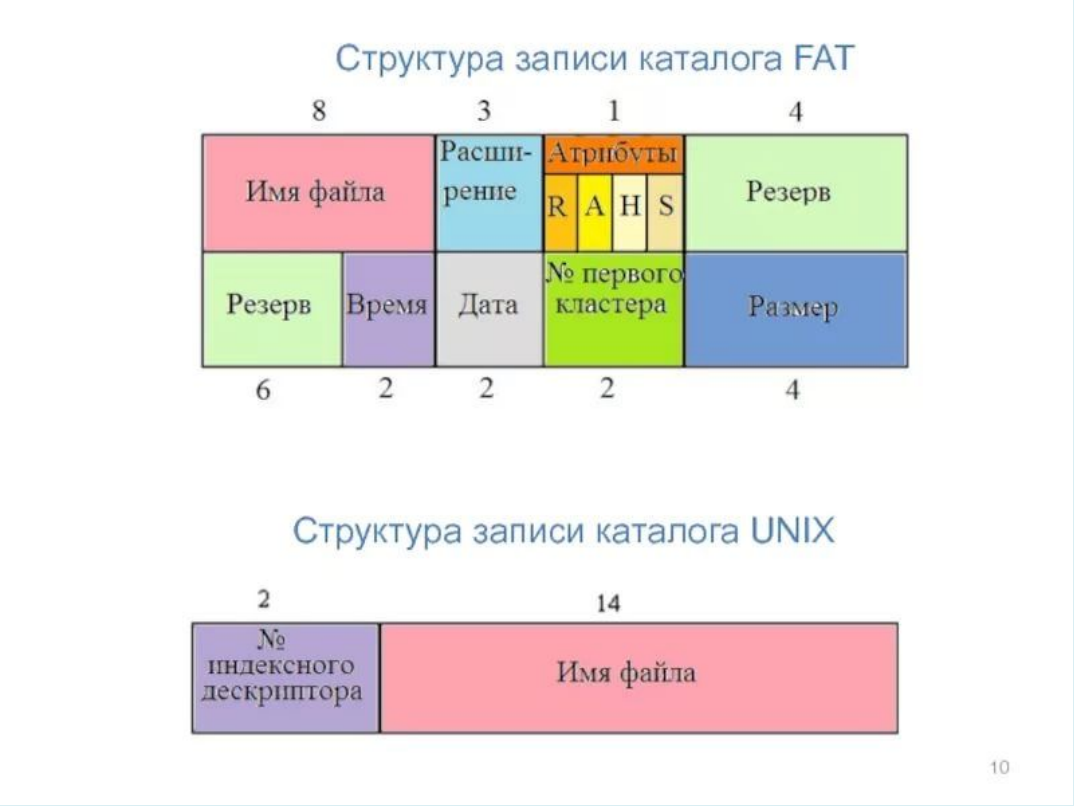
Определить какие латинские буквы встречаются в заданной строке

```
Program Pr1;
Type T=set of ‘a’..’z’;
var
Stroka: string;
Lchars:T;
Simv: char; i: byte;
Begin
Readln (stoka);
Lchars:=[];
      For I:=0 to Length (stroka) do
            If stroka[i] in [‘a’..’z’] then Lchars:=Lchars+stroka[i];
      For simv:=‘a’ to ‘z’ do
            If simv in Lchars then
Write (simv, ‘ ‘);
End.
```

Значение множественного типа просто так нельзя вводить и выводить. **Можно ввести** значение элементов множества и добавить их к множеству, используя операцию объединения.

Для вывода необходимо организовать цикл, в котором проверяют вхождение во множество всех элементов базового типа и выводят только те, которые относятся к нашему множеству.

Тема 11. **Базовая структура данных - записи**



Запись – это структура, позволяющая хранить в себе разнотипную информацию, связанную по смыслу.

Например:

Запись, содержащая информацию о **студентах**: *номер группы, телефон, дату рождения, фамилию.*

№	Фамилия	Адрес	Группа
1	Петров	Казань	ПИ-012
2	Иванов	Москва	ГД-013

Для описания записи в программе в разделе типов указывается:

- Имя всего типа записи
- Имена полей, из которых будет состоять данная запись
- Для каждого поля записи указывается тип хранящейся в ней информации.

Количество полей в записи может быть **любым**, все зависит от задачи.

Общий вид:

Type *имя_записи* **= *record***
Имя_поля1: тип;
Имя_поля2: тип;
…
Имя_поляN: тип;
End;

Пример 1

Описать запись, которая содержит информацию о студентах группы: фамилию, адрес, номер группы.

```
Program Pr1;
Type student = record
        Fam: string;
        Adres: string;
        Gruppa: byte;
End;
Var st: student;
Begin
        St.fam:=‘Иванов’;
        St.adres:=‘Казань’;
        St.gruppa:=‘p-221’;
```

Пример 2

Предположим, что нам необходимо хранить информацию о студентах из 25 человек. В этом случае, мы можем описать массив записей.

```
Program Pr2;
Type student = record
        Fam: string;
        Adres: string;
        Gruppa: byte;
End;
Var st: array [1..25] of student;
```

Необходимо для пятого студента ввести фамилию – Иванов

```
St[5].fam:=‘Иванов’;
```

Пример 3

Описать запись, содержащую информацию о сотрудниках организации: фамилию, должность, оклад. В организации работают 25 человек. Ввести необходимую информацию и вывести информацию на экран.

```
Program Pr3;
Const m=25;
Type sotrydniki = record
        Fam, doljnost: string[30];
        oklad: integer;
End;
Var sotr: array [1..25] of sotrydniki;
i:integer;
Begin
Writeln (‘введите сотрудников организации’);
      For i:=1 to m do begin
      Writeln (‘введите фамилию’ , I, ‘сотрудника’);
      Readln (sort[i].fam);
      Writeln (‘введите должность’ , I, ‘сотрудника’);
      Readln (sort[i].doljnost);
      Writeln (‘введите оклад’, I, ‘сотрудника’);
      Readln (sort[i].oklad);
      End;
For i:=1 to m do begin
     Writeln (‘сотрудник’ , i);
     Writeln (‘фамилия:’ , sort[i].fam);
     Writeln (‘фамилия:’ , sort[i]. doljnost);
     Writeln (‘фамилия:’ , sort[i]. oklad);
     End;
end;.
```

Если в программе используется переменная записи, состоящая из нескольких полей, ввод и вывод информации в данную запись осуществляется отдельно по конкретным полям.

```
Type sotrydniki = record
        Oklad: integer;
        Fam, address : string
End;
Var s1, s2: sotrydniki;
Readln( s1.fam);
Readln (s1.address , s1.oklad);
S2:=s1;
```

Значение переменной записи в программе можно присваивать, это значит, что значения полей s2 станут значения соответствующих полей записи s1.

В качестве типа полей записи могут использоваться любые существующие стандартные типы, а также массивы.

Пример 4

Необходимо сформировать запись о студенте группы со следующей информацией: фамилия, год рождения, оценки. Для студента может быть 10 оценок.

```
Program Pr4;
Const m=10;
Type student= record
        Fam:string;
        god: integer;
        Ocenki: array [1..m] of byte;
End;
Var
St: student;
Begin
     St.fam:='Orlov';
     St.ocenki[5]:=2;
End;
```

Пример 5

Создать запись о студентах группы, содержащую следующую информацию: фамилию и его оценки за семестр. Для каждого по 5 оценок. Всего в группе 25 студентов. Оформить программу с помощью пользовательского меню:

- Внести информацию о студентах;
- Вывести на экран все сведения о студентах
- Поиск студентов по фамилии с выводом информации

```
Program Pr5;
Const n=25; m=5;
Type Student = record
        fam: string [30];
        Ocenki: array [1..m] of byte;
End;
Var st: array [1..n] of student;
      I, j, k, p: integer;
      F: string[30];
Begin
Repeat
Writeln (‘1- ввод информации’);
Writeln (‘2 – вывод информации’);
Writeln (‘3 – поиск студента по фамилии’);
Writeln (‘4 - выход’);
Readln (k);
Case k of
1: for i:=1 to n do begin
        Writeln (‘введите фамилию’ , I, ‘студента’);
        Readln (st[i].fam);
        Writeln (‘ введите ‘, m, ‘оценок’);
            For j:=1 to m do
              Readln (st[i]. ocenki[j]);
End;



2: for i:=1 to n do begin
```

При работе с записями встречается понятие **вложенной записи** – это, когда в качестве поля одной записи выступают тоже записи

Пример:

Оформим запись сотрудников организации, содержащую: фамилию, должность, дату поступления на работу, при этом дата поступления должна отображать число, месяц, год. В этом случае удобно для поля записи о сотруднике использовать вложенную запись, состоящую из трех полей : число, месяц, год.

```
Type D =record
        Day: byte;
        Month: string[15];
        Year: integer;
End;
Type sort=record
        Fam, doljnost: string[50];
       data: D;
End;
Var s:sort;
Begin
        s.fam:=‘Пупочкин’;
        s.data.day:=15;
        s.data. Month :=‘май’;
        s.data. Year :=2020;
```

Для удобства работы с вложенными записями в программе нужно использовать специальный оператор из имени записи **With…Do**

```
 With s do Begin
                fam:=‘Пупочкин’;
                data.day:=15;
                ata. Month :=‘май’;
                data. Year :=2020;
            End;
```

Тема 12. **Использование файлов в программе**

Общие понятия об использовании файлов

Файлы могут использоваться для хранения информации, обрабатываемой программой.

Любая обработка, хранящейся в файле , осуществляется **только после загрузки** ее в ОП. Созданные в программе данные можно записать в файл при запуске программы можно и прочитать с файла.

Файл – это последовательность компонентов, относящихся к одному типу: файл целых чисел, записи, строк.

Доступ компонентов к файлу осуществляется через **указатель файла**, когда мы читаем или записываем файл, этот указатель автоматически перемещается на следующие компоненты. Для идентификации файлов в программе используют **файловые переменные**.

В зависимости от информации различают 3 типа файлов:

1. Типизированные файлы

Содержат последовательность однотипных компонентов, тип которых заранее указываются в описании. В качестве компонентов могут использоваться любые из существующих типов, а также существующие структуры, т.е. записи.

Type *имя_файла* **= *file of*** *тип_компонентов;*

Type Fint **= file of** integer;

FR = **file of** real;

2. Текстовые файлы

Компонентами текстового файла являются строки переменной длины.

Type *имя_файла* **= *text****;***

Type TxtFile **= text;**

3. Нетипизированные файлы

Type *имя_файла* **= *file;***

Type FF **= file;**

Со всеми видами файлов в программе можно выполнять следующие действия:

- *Описание файлового типа данных.*

Имя файла

- Объявление необходимого количества специальных переменных - файловых переменных.

В программе число файловых переменных должно соответствовать числу одновременно использования в программе файлов.

```
var f1, f2, f3: file of integer;
F4,f5: TxtFile;
```

- Инициализация файла

В первую очередь в теле программы необходимо выполнить **инициализацию файлов**, т. е. установить связь описанной файловой переменной в конкретном файле на диске. Для этого используется специальная подпрограмма **Assign.**

При этом, если мы указываем просто имя файла, то он должен храниться в каталоге.

Полное имя: C:\Temp\Myfile.txt

Пример:

```
Var f1: file of integer;
Name: string;
Begin
    Assign (f1, ‘data’);
    Assign (f1, D:\Temp\Myfile.int);
```

После выполнения инициализации файловой переменной , и дальше в программе имена дисков **не используется**.

Имя файла можно задавать программно или спрашивать у пользователя перед выполнением инициализации.

```
Name: string;
Writeln (‘введите имя файла’);
Readln (name);
Assign (f1, name);
```

- Открытие файла – подготовка файла для выполнения операции ввода или вывода.

Для открытия типа для записи используется подпрограмма **Rewrite**

Rewrite (файловая_переменная);

- Если файла с заданным именем нет в указанном каталоге, то он автоматически создается, как пустой и подготавливается для записи.
- Если файл уже существует и содержит какие-либо данные, то он полностью очищается без каких-либо предупреждений, а также становится готовым для новой записи.

Для подготовки файла для чтения используется подпрограмма **Resert**

Reset (файловая_переменная);

- Если открывающий файл для чтения не существует, то возникает программная ошибка.

  Run Time Error
- Если файл существует, то можно из него читать информацию.

Обработка компонентов файла

Работа с файлами выполняется с помощью инструкции цикла. Основные операции – **запись и чтение**.

Для записи используется инструкция

Write *(файловая_переменная, список_переменных);*

Чтение из файла

При чтение количество компонентов в файле заранее неизвестно. Поэтому чтение осуществляется **до достижения конца этого файла**. С каждым файлом связана специальная внутренняя переменная указатель текущей позиции. При открытии файла этот указатель автоматически определяет первый элемент файла. После чтения он автоматически перемещается к следующему элементу файлу и так до конца файла.

При достижение указателем конца файла он устанавливает значение специальной функцией **EOF (End of File).** Функция может иметь значения: истина или ложь. Она устанавливается истиной, если указатель дошел до конца.

While not EOF *(файловая_переменная)* ***do*** *чтение;*

Само чтение элементов из файла выполняется инструкцией

Read (*файловая_переменная, список_переменных****);***

6. Закрытие файла. После завершения работы с файлом используется подпрограмма

Close (*файловая_переменная****);***

После закрытия файла можно снова его инициализировать с помощью подпрограммы **Reset** или **Rewrite**.

Типизированные файлы

```
Type Fint = file of integer;
Var f1, f2, f3: Fint;
    X1, x2, x3: integer;
Read (файловая_переменная, список_переменных);
Reset (f1);
Read (f1, x1);
Write (f2, x2);
```

Пример:

```
Type Fint =f ile of integer;
Var f1, f2, f3: Fint;
    X1, x2, x3: integer;
    Name1: string;
Begin
Writeln (‘введите имя файла’);
Readln (nam1);
Assign (f1, name1);
Rewrite (f1);
Writeln (‘введите число’);
Readln (x1);
Write (f1, x1);
Close (f1);
End.
```

Пример 1

Написать программу, которая генерирует 10000 случайных чисел, записывает их в файл, а затем читает с выводом на экран.

```
Program Pr1;
Type Tfint = file of integer;
Var fpint:Tfint;
      I, n : integer ;
Begin
Assign (Fpint, 'MyInt.dat');
Rewrite (Fpint);
     For i:= 1 to 10000 do begin
            N:=Random (1000);
           Write (fpint, n);
     End;
Close (Fpint);
Reset (Fpint);
      While not EOF (Fpint) do
       Begin
             Read (Fpint, n);
             Write (n:4);
        End;
Close (fpint);
End.
```

Пример 2

Копирование из одного числового файла в другой.

```
Program Pr2;
Var FromF, ToF = file of real;
      N: real;;
      FrName, ToName: string;
Begin
Radln (FrName);
Readln (ToName);
Assign (FromF, FrName);
Assign (ToF, ToName);
Reset (FormF);
Rewrite (ToF);
      While not EOF (FromF) do
       Begin
            Read (FromF, n);
            Write (ToF, n);
       End;
Close (FromF);
Close (ToF);
End.
```

Пример 3

Создать 2 файла, содержащих по 50 целых чисел, сгенерированных в диапазоне от 0 до 99. Прочитать эту информацию с файла с выводом на экран. Создать третий файл, полученный в результате объединения этих двух файлов. Объединение файлов последовательно. Прочитать информацию с результирующего файла с выводом на экран.

```
Program Pr2;
Var f1, f2, f3: file of integer;
      I, k : integer;
      N1, n2, n3: string[50];
Begin
Writeln (‘введите имя файла 1’);
Readln (n1);
Writeln (‘введите имя файла 2’);
Readln (n2);
Assign (f1, n1);
Assign (f2, n2);
Rewrite (f1);
      For i:= 1 to 50 do begin
            k:=Random (100);
            Write (f1, k);
      End;
Close (f1);
Rewrite (f2
      For i:= 1 to 50 do begin
      k:=Random (100);
      Write (f2, k);
     End;
Close (f2);
Reset (f1);
      While not EOF (f1) do
            Begin
                  Read (f1, k);
                  Write (k:4);
              End;
Close (f1);
Reset (f2);
```

Объединение выполнить по-элементно. *(Фрагмент объединения)*

```
 Reset (f1);
Reset (f2);
Rewrite (f3);
          While  not EOF (f1)  and not EOF (f2) do
                   begin
                         Read (f1,k1);
                         Write (f3, k1);
                          Read (f2,k2);
                          Write (f3, k2);
                    End;
Close (f1);
Close (f2);
Close (f3);
End.
```

Файл записи

В качестве компонентов типизированного файла могут выступать структуры записи.

Для работы необходимо:

- Сначала описать саму запись
- Описать типизированный файл, компонентами которого будет являться описанная структура записи.
- В разделе описания ввести переменную типа запись, чтобы была возможность в программе записывать и читать из файла компоненты-записи.

Чтобы записать или прочитать из файла компонент, используется **полностью сформированная** запись (а не по отдельным полям).

Пример 4

Программа создания и чтения файла записи. Сформировать запись о сотрудниках организации, содержащую следующую информацию: фамилию, адрес, оклад сотрудника.

Создать файл, содержащий 20 записей о сотрудниках организации. Прочитать всю информацию с выводом на экран.

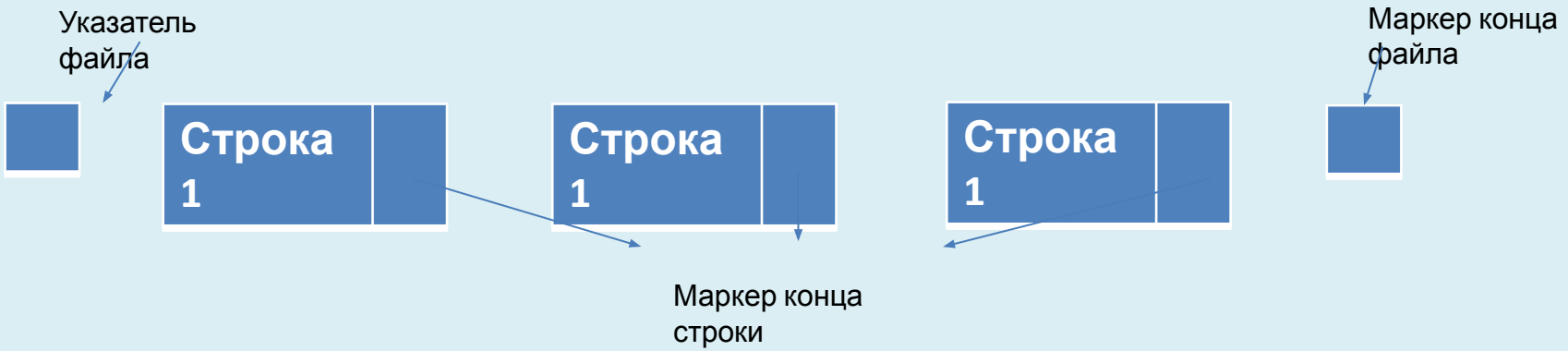
```
Program Pr4;
Type Sotrydniki = record
        Fam, address: string [50];
        Oklad: integer;
End;
FF= file of sotrydniki;
Var f1:FF;
      Name: string;
      S: array [1..20] of Sotrydniki;
      i: integer;
Begin
Writeln (‘введите имя файла’);
Readln (name);
Assign (f1, name);
Rewrite (f1);
     For i:=1 to 20 do
          begin
               Writeln (‘имя’);
                Readln (s.fam);
                Writeln (‘адрес’);
                Readln (s.address);
                Writeln (‘оклад’);
           Readln (s.oklad);
```

Текстовый файл

Текстовый файл – это набор строк переменной длины. Обработка текстового файла осуществляется по-строчно.

Для обозначения конца строки в файле автоматически записываются так называемые указатели конца и начала строки – это символы CR[13], LF[10].

Описание: Var FT: text;



Запись в текстовый файл

Writeln (файловая_переменная, текстовая_строка);

Например:

```
Var ft: text; st:string;
Writeln (ft, ‘пример1’);
Wtiteln (‘введите строку для записи в файл’);
Readln (st);
Writeln (ft, st);
```

Чтение из файла:

```
Read(ft, st);
```

Пример 4

Создать текстовый файл, содержащий 10 текстовых строк, введенных пользователем с клавиатуры. Прочитать и вывести на экран.

```
Program Pr1;
Var ft: text;
    Name, st: string;
    i:integer;
Begin
Writeln (‘введите имя файла’);
Readln (name);
Assign (ft, name);
Rewrite (ft);
      For i:=1 to 10 do begin
            Writeln (‘введите’, I, ‘ строку’);
            Readln (st);
            Writeln (ft, st);
      End;
Close (f1);
Reset (f1);
      While not EOF (Ft) do begin
            Readln (ft, st);
            Writeln (st);
     End;
Close(f1);
end.
```

Использование текстовых файлов для хранения числовой информации

Преимущество:

Данные файлы можно создавать, просматривать и редактировать с помощью обычных текстовых редакторов.

Минус:

Любой текстовый файл с числовой информацией будет больше по сравнению с типизированным файлом.

Каждое число **integer** занимает 4 байта. В текстовом файле числовая информация хранится в виде символов. Каждый символ занимает по 1 байту.

Число -12345 , в типизированном файле будет занимать 4 байта, а в строковом 6 байт.

Операции: запись и чтение в файл.

Запись:

Чтобы записать информацию используется

Write (файловая_переменная, список_перемменных);

Пример:

Опишем в разделе описания

```
Type TF: text;
Var f1:TF; x:integer;
Begin
Assign (f1, ‘пример’);
Rewrite (f1);
Write (f1, x:4);
Close (f1);
```

При записи число в текстового файл необходимо сразу указывать **формат** записываемого числа.

Пример:

Записать в текстовый файл 1000 целых чисел при этом число записать в виде таблицы 50x20. Для указания при записи признака окончания строки используется подпрограммы Writeln (f1)

```
Rewrite (f1);
    For i:=1 to 50 do begin
        For j:=1 to 20 do begin
            X:=random(100);
            Write (f1, x:4);
        End;
    Writeln (f1);
     End;
Close (f1);
End.
```

Чтение из файла

Read (f1,x);

Вложенный файл

EOF (до конца файла)

EOLN (до окончания строки)

Чтение новой строки: **Readln** (f1);

```
Reset(f1);
While not EOF (f1) do Begin
                While not EOLN (f1) do Begin
                                Read (f1, x);
                                Write (x:4);
                 End;
Readln (f1);
Writeln;
End;
Close (f1);
end.
```

Пример 5

Создать текстовый файл из 15 строк, введенных пользователем с клавиатуры. Прочитать его с выводом на экран. Создать 2-ой файл текстовый, представляющий из себя копию первого файла.

```
Program Pr5;
Var f1, f2: text;
Name1, name2, st: string;
i: integer;
Begin
Writeln (‘введите имя 1ого файла’);
Readln (name1);
Assign (f1, name1);
Rewrite (f1);
      For i:=1 to 15 do begin
            Writeln (‘введите текст’);
             Read (st);
             Writeln (f1, st);
      End;
Close (f1);
Reset (f1);
While not EOF (f1) do begin
          Readln (f1, st);
          Writeln (st);
          End;
Writeln (‘ введите имя второго файла”);
```

Тема 13. **Использование в программах подпрограммы**

ПРОЦЕДУРА И ФУНКЦИИ

Подпрограммы – это набор логически связанных инструкцией программы, которая оформлена специальным образом, обязательно имеет имя, может многократно использоваться в программе.

Структура подпрограммы:

- Любая подпрограмма имеет заголовок с именем
- Раздел описания
- Раздел выполняющих инструкций

Фрагмент основной программы:

- **Главная программа** – это набор инструкций, с которых начинает выполнятся программа при ее запуске.
- Набор вспомогательных подпрограмм, которые могут вызываться из главной программы и других программ.

Количество использованных подпрограмм может быть **любым**. При этом подпрограммы могут вызываться и внутри других подпрограмм этой программы.

Для использования реализованных в подпрограмме действий, выполняется ее вызов или обращение.

При вызове в подпрограмме достаточно в нужном месте указать ее **имя** (в простейшем случае).

При этом одну и ту же подпрограмму мы можем вызывать **любое количество раз** в том числе и циклически.

Та программа, к которой обращаются называют **вызывающей**, которую вызывают называется **вызываемой**.

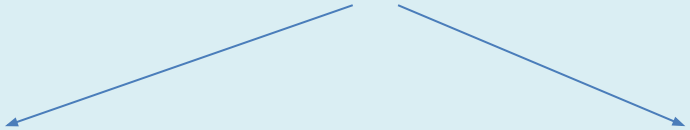
Порядок взаимодействия:

- При вызове любой подпрограммы, выполнение вызывающей подпрограммы приостанавливается.
- Процессор начинает выполнять действия, вызванной подпрограммы.
- После выполнения всех действий подпрограммы возобновляется работа основной программы с того места, где она была приостановлена.

Преимущества:

- Возможность многократного использования однажды созданного кода (при этом использовать его с различными исходными данными).
- Упрощение понимания структуры основной программы, так как она будет представлена в виде взаимодействующих подпрограмм.

Используются подпрограммы 2 видов:



Процедуры *Функции*

Процедуры

Общий вид:
Procedure *имя_процедуры;*
Const..
Type..
Var..
Begin
.. //Инструкции тела подпрограммы..
End;

Пример 1

```
Procedure P1;
Const n=5;
Var i:integer;
Begin
     For i = 1 to n do
     Writeln (‘привет’);
End;
```

Пример 2

```
Program Pr2;
Var….
Procedure имя;
Var ..
Begin
…
End;
Begin //основная программа)
…
///имя процедуры
p1;
…     End.
```

Пример 2

Реализовать программу, использующую процедуру, осуществляющих вывод в виде линий.

```
Procedure Pr1;
    Procedure p1;
       Var i: byte;
        Begin
            For I:=1 to 25  do write (‘*”);
            Writeln;
            End; // конец процедуры
Begin //начало основной программы
P1;
Writeln (‘Linia1’);
P1;
Writeln (‘Linia2’);
P1;
P1;
End.
```

Функция

Общий вид:

Function *имя_функции: тип_возвращаемой_подпрограммы;*

Раздел описания

Begin

… //выполнение инструкции

…

Имя_функции:=значение; или result:= значение;

End;

В отличие от процедур, после выполнения функции мы получаем **1 результат**, значение которого мы должны занести в имя функции. Поэтому при описании функции в заголовке обязательно указывается **тип возвращаемого результата**.

А в теле самой функции после выполнения всех необходимых инструкций, обязательно нужно выполнить инструкцию присваивания – в имя функции ***занести полученный результат***.

И вызов функции в основной программе будет отличаться от вызова процедур. !!!**Нельзя просто написать имя функции.**

В программе мы можем пользоваться именем функции, *как переменной* того типа, который был указан в заголовке, в который был указан результат

При использовании подпрограмм возникают понятия, как глобальные и локальные переменные.

Глобальные – это переменные, которые описаны в разделе описания основной программы. Их мы можем использовать как в основной программе, так и в подпрограмме.

Локальные – это переменные, описанные в подпрограмме. Они используются только внутри подпрограммы.

Понятия локальности и глобальности переменных является относительным, т.к. внутри процедур могут быть организованы внутренние процедуры.

```
Program Pr;
Var I, j: integer;
...
Procedure P1;
Var  k:integer;
   Procedure P2;
    var y: integer;
```

I, j – глобальные, их мы можем использовать и в процедурах P1, P2

K – локальная по отношению к P1б но в то же время может использоваться как глобальная переменная внутри процедуры P2

Y – это локальная переменная и может использоваться только в процедуре P2.

Использование одноименных локальных и глобальных переменных

```
Program Pr;
Var k: integer;
    Procedure P1;
    Var k: integer;
    Begin
     ...
    K:=15;
    End;
Begin
...
K:=10;
P1;
Writeln (k);
End;
```

При использовании в программе локальных и глобальных переменных, значение локальных переменных **перебивает** значения глобальных переменных.

Пример 4

Разработать программу, которая позволяет:

1. Генерировать и выводить на экран массив из 100 целых чисел от -50 до 50.
2. Осуществлять поиск максимального и минимального элементов в этом массиве.
3. Подсчитывать среднее значение всех элементов с выводом на экран.

Оформить программу с использованием пользовательского меню. Каждое действие оформить как отдельные подпрограммы. Нахождение среднего значение оформить в видефункции, нахождение максимального и минимального значения в виде процедуры

```
Program Pr4;
Const n = 100;
Var a: array [1..n] of integer;
Min, max, I, k: integer;

Procedure P1;
Begin
Randomize;
For i:=1 to n do begin
A[i]:=random(101)- 50;
Write (a[i]:4);
End;
Writeln;
End;

Procedure p2;
Begin
Max:=a[1];
Min:=a[1];
For i:=2 to n do begin
      If a[i]> max then max:=a[i];
       If a[i]<min then min:=a[i];
End;
End;

Function f1: real;
Var sum: integer;
Begin
sum:=0;
      For i:=1 to n do
```

Формальные и фактические параметры (входные и выходные)

При взаимодействии подпрограмм и основной программы, часто бывает необходимым вызывать подпрограммы с какими-нибудь входными данными. Это удобно использовать, когда одну и ту же подпрограмму, мы хотим использовать в программе при разных вызовах с различными исходными программами.

Например:

Можно оформить подпрограмму генерации и вывода массива на экран, тогда использование размера массива и диапазона генерации ,как входной параметр позволит, как внутри одной программы при помощи одной и той же процедуры генерировать различные массивы.

```
Program Pr;
Const  = 100;
Var a: array [1..n] of integer;
      i: integer;
      Procedure P;
      Begin
          For i:=1 to n do begin
                A[I]:=random (101)-50;
                Writeln (a[i]:4);
                End;
       Writeln;
        End;
Begin
…
P;
…
P;
…
End.
```

Использование входных параметров

При использование подпрограмм, кроме входных можно применять и **выходные параметры**, т.е. те значения, которые получаются в результате выполнения в соответствующей подпрограмме и в дальнейшем, которые используются в основной программе.

Выходные параметры также описываются в заголовке, соответствующей процедуры или функции (после ее имени) , выходной параметр описывается в скобках после директивы var.

Например:

***Procedure** P(x, y:integer; **var** k: integer; z :real);*

…

P (10, 50, m, n);

Пример 5

Оформить процедуру, позволяющую вывести на экран разделяющую линию из любого символов количество и типов.

```
Procedure P1 (s: char; k: integer);
Var I:integer;
Begin
For i:=1 to k do
Readln (s);
Writeln;
Begin {основная программа}
…
P1 (‘*’ , 50);
P1  (‘ $’, 100);
…
Writeln (‘введите символ’);
Readln (a);
Writeln (‘вызов количества’);
Readln (kol);
P1 (a, kol);
End.
```

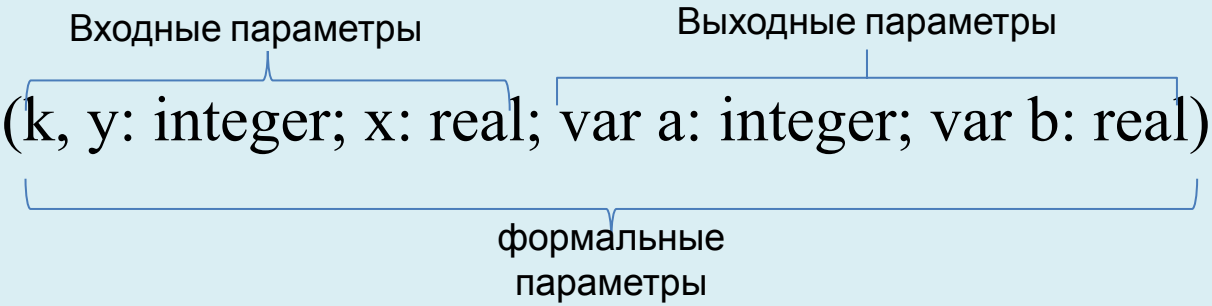
Формальные параметры

Фактические параметры

Параметры переменные и параметры значения

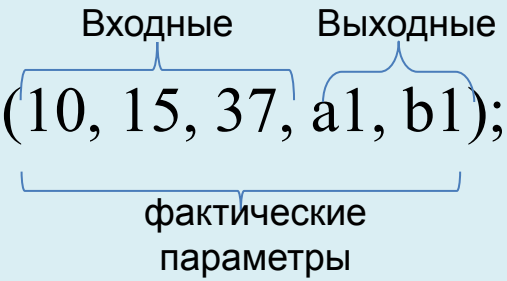
Часто встречаются случаи, когда в качестве результата выполнения подпрограмм получается несколько заголовков. В этом случае удобно оформить с использованием формальных параметров, которые делятся на разновидности:

- Входные параметры (параметры переменные)
- Выходные параметры (параметры значения)



.....

```
Var a1: integer;
    b1: real;
```



Если формальный параметр объявлен **БЕЗ** директивы **Var**, то в подпрограмму передается **КОПИЯ** соответствующего фактического значения, и поэтому любые изменения формального параметра в подпрограмме никак НЕ изменяют фактическое значение.

Наоборот, если формальный параметр объявлен с директивой **var**, то в подпрограмму передается **АДРЕС** соответствующей фактической переменной, т.е. подпрограмма на самом деле непосредственно работает с самой фактической переменной.

Пример 1

Реализовать поиск максимального элемента и его номера в заданном подмассиве. Поиск оформить с использованием процедуры с входными переменными (левая, правая границы). Массив из 100 чисел [0, 99].

```
Program P1;
Const m=100;
Var a: array [1..m] of integer;
      i, l, z, max, nom: integer);
Procedure P1 (pl, pr: integer; var pmax, pnom: integer);
Begin
Pmax:=a[pl];
Pnom:=pl;
      For i:= pl to pr do
          If a[i] > pmax then
          begin
              Pmax:=a[i];
              Pnom:=i;
          End;
End;

Begin
Randomize;
For i:=1 to m do
      begin
          A[i]:=random (100);
```

Рекурсивные подпрограммы

Рекурсивная программа – это программа, которая в своем теле вызывает сама себя. При этом важно помнить, что любую рекурсивную программу можно заменить на обычную с использованием циклов, т.е. выполнить ее итеративно.

Схематично рекурсивная программа выглядит следующим образом

*Procedure **P1**;*
Begin
...
P1;
...
End;

Рекурсивные вызовы **не должны** продолжаться **до бесконечности**. Для этого в рекурсивной подпрограмме вводят формальные параметры – значения, которые при рекурсивном вызове уменьшаются, а внутри самой рекурсивной программы выполняется проверка значений этих переменных.

Схема основной программы с рекурсивной подпрограммой:

```
Program P1;
Var …
    Procedure Recurs (k: integer);
      Var …
        Begin
        If  k>0 the recurs (k-1)
```

Механизмы взаимодействия вызывающей программы и вызываемой

Рекурсивная подпрограмма всегда имеет формальный параметр, поэтому рекурсивный вызов самой себя, приводит к ситуации одноименных переменных, вызывающей и вызываемой программы.

В этом случае внешние одноименные переменные становятся не доступны в вызванной подпрограмме и при этом ее значение сохраняется до тех пор, пока не завершится вызванная подпрограмма.

После завершения подпрограммы недоступными становятся внутренние переменные в силу своей локальности, а внешние наоборот могут использовать те значения, к которым она имела до вызова подпрограммы.

Ситуация становится еще более интересной, когда внутри вызванной подпрограммы производится вызов своей внутренней подпрограммы с таким же одноименным параметром: к заполненному ранее значению добавляется еще одно на время работы самой внутренней подпрограммы с восстановлением запомненных значений в обратом порядке.

Такой механизм запоминания называется **стековым механизмом**.

Стек – это линейная структура, которая элементу добавляет с одного конца (вершина стека) и извлекается из вершины. Элемент, который вошел последним, извлекается первым. Раньше восстанавливается значение того параметра, который был заполнен позже всех.

При завершении работы, происходит восстановление из стека последнего заполненного значения.

Пример 2

Ввести исходное значение.

1. Проверяем входной формальный параметр на нуль.
2. Выводим текущее значение
3. Программа рекурсивно вызывает сама себя со значением параметра на 1 меньше текущего.
4. После завершения очередного рекурсивного вызова и возврата в вызывающую подпрограмму, выводит восстановление значения формального параметра.

```
Program Pr1;
Var k: byte;
    Procedure Rec (k:byte);
    Begin
        If k=0 then Writeln (k: 3, ':' , 'вызов закончен начинаем возврат')
            Else begin
                    Write (k:3);
                     Rec (n-1);
                     Write (n:3);
                   End;
    End;
Begin
Readln (k);
Rec (k);
End.
```

Пример 3.

Программа рекурсивного вычисления факториала.

```
Program Pr3;
Var n, fact : integer;
      Function f(n:integer): integer;
      Begin
          If n> 0 then f:= n* f(n-1) else
              F:=1;
       End;
Begin
Readln (n);
Fact:=f(n);
Writeln (‘fact =‘, fact);
end.
```